BASICS OF THE RENESAS SYNERGY™ PLATFORM

Richard Oed

RENESAS synergy™
Accelerate. Innovate. Differentiate.

BIG IDEAS
FOR EVERY SPACE

2017.03

BASICS OF THE RENESAS SYNERGY™ PLATFORM

Richard Oed

GUIDANCE ON USING THE 'BASICS OF THE RENESAS SYNERGY™ PLATFORM' BOOK

The Synergy Basics Book is intended to be used as both an introductory guide for engineers taking their first steps with Synergy, as well as a heavily indexed reference book for those already familiar with the Platform. Please be informed that this version of the book refers to the SSP release v1.2.

As the Platform is being continually developed it's inevitable that by the time we go to print, there will be sections which are no longer up to date. To ensure the book remains useful for the above purposes, there is a supporting online ecosystem which you can find at www.renesas.com/synergy-book. In the book we identify three different types of information – information which will not change (such as the introductory chapters), information which will be maintained (such as software examples) and information which is only published online (such assections that would otherwise be repetitive and un-necessarily consume space in the book).

CONTENTS

FOREWORD

If you are really lucky then, once in a lifetime, something will happen that changes your life or the lives of those around you. I've been fortunate enough to experience that twice. Maybe even three times! I don't mean in the sense of winning the lottery, or some other personal experience, but rather in my work life.

The first experience was when I joined Hitachi in April 1989. Hitachi was proceeding with a controlled withdrawal from manufacturing 63xx based microcontrollers which had been made under license from Motorola until the license agreement was withdrawn, for reasons I'm not party to. That isn't the story though. When it was clear that we would have to inconvenience customers by stopping the manufacuring of a popular MCU line up, Hitachi's response was to hire a large team of engineers and task them with designing a completely new, next generation of microcontrollers. What followed retained the key benefit of the 63xx families they succeeded, namely electrical compatibility between OTP (one time programmable) and mask devices. This made prototyping much simpler than before when you had to wait for working samples from the mask production before you could start evaluating prototypes. And, at a time when engineers were deserting complex assembly language for the convenience of higher level language programming techniques, they introduced a register based architecture that enabled much more efficient interrupt processing and hence code efficient and fast to execute structured programming. Followed in the early 1990s with a world beating flash technology which to this date has never seen a field failure that we are aware of.

The H8 delivered so many benefits it became the MCU of choice in many applications. At its peak in the late 90s we were delivering more than 1 M pcs per month for use in motor control in PC drives. They were the microcontroller of choice for one of the earliest smart meter programmes to which we shipped more than 30 M devices alone. And many many other applications. That it achieved such wide acceptance was to confirm the benefits it offered for engineering communities. Did it change their lives? Maybe that's too strong a claim. But it certainly enabled designers to do things previously not possible.

OK. I mentioned two other life changing innovations. The next was the invention of TFT colour LCD displays. I still remember calling customers and offering "a demonstration of a new colour display which offered unrivaled high contrast, unbelievable viewing angles and…full colour!" There was no customer refusing a visit! I was fortunate to secure the first customer design-in in Europe with whom I made my first business trip to Japan to see TFT displays being produced. That's what can happen when you deliver life changing technology, products and solutions.

In the years that followed, Hitachi merged with the semiconductor businesses of Mitsubishi and latterly NEC, both of whom possessed great MCU line ups in their own right. Once again we had some incredible claims...the world's largest MCU company by a long long way, our own flash technology offering the highest performance (zero wait state access at 120 MHz), a 40 nm process enabling exceptional integration (up to 8 MB on chip flash) and our own 300 mm manufacturing lines ensuring the highest quality.

Around the same time, many other silicon vendors stopped investing in their own CPU core developments and started licensing ARM® technology for general purpose MCUs. Gradually customers started choosing what they perceived as a "standards based" ARM® core rather than proprietary cores, regardless of the benefits the latter could offer. We could write a lot of text on whether ARM® based microcontrollers are truly standards based, but let's leave that for another place. The fact was that more and more customers requested them. Some would even say ARM® commoditised the market, driving prices to historical lows. Great for customers, but not so good for a company with microcontrollers as a core business. It was time for a rethink. We talked to customers about better proprietary cores, but most applications didn't require higher performance. We talked about more integration, but the majority of systems had enough memory and peripheral options available on chip. Maybe this was our first clue that the future didn't lie exclusively in the hardware. But we didn't recognise it at the time.

In the early days of 2013, I was assigned to a small team in Renesas Electronics Europe to discuss our future microcontroller strategy. It was clear something had to change. The inclusion of ARM® cores was almost inevitable due to the growing clamour from customers. But we knew this would change nothing on its own...there were already more than ten other ARM® MCU vendors in the market (a number that has recently reduced substantially due to a number of mergers & acquisitions). We had to find something completely innovative.

During our subsequent meetings we consulted with a number of key customers about the daily challenges of their design teams. What became clear was that the microcontroller was not going to be the only focus in the future. The designer's challenge was now in software, test + integration, communications, security ... . Hastened by the spectre of the Internet of Things.

So our project began. And what a project! Not only designing the most intercompatible families of ARM® based MCUs, but also a massive software platform comprising over 1 M lines of code and representing some 200 man years of development time, a truly convenient toolchain and many other innovations. We call it the Renesas Synergy™ Platform. It's a game changer. You can start programming at the API, reduce your time to market, save money along the way and spend time on what you do best – innovating!

That's what this book is about ... helping you take your first steps with the Renesas Synergy™ Platform. It might just change your life too!

Andy Harding
Director, Core Marketing
Renesas Electronics Europe

1 INTRODUCTION TO THE RENESAS SYNERGY™ PLATFORM

When I was asked to author a book to help engineers take their first steps with the Renesas Synergy™ Platform, I was at first honoured and excited, but a short time later, quite daunted. *Where on earth to begin?* Put simply Renesas has achieved a revolutionary new level of abstraction in the design process, enabling the designer to start development at the Application Programming Interface (API), thus reducing time to market, lowering total cost of ownership and removing barriers to entry. Beneath that API layer sits a massive software platform which represents more than 200 man years of development work and over a million lines of code. And Renesas is prepared to support, warrant and maintain the complete platform as a product, to their usual high standards of quality and reliability.

So what exactly is the Synergy Platform? Figure 1-1 shows the five main elements of the Synergy Platform:

- **Synergy Software:** A complete product-quality software platform with common APIs.
- **Synergy Microcontrollers**: A family of scalable, ARM® Cortex®-M based microcontrollers (MCUs).
- **Synergy Gallery:** Web access to Synergy specific software, tools, and more.
- **Synergy Tools and Kits:** Intuitive development tools and kits.
- **Synergy Solutions:** Specific solutions for products and applications.
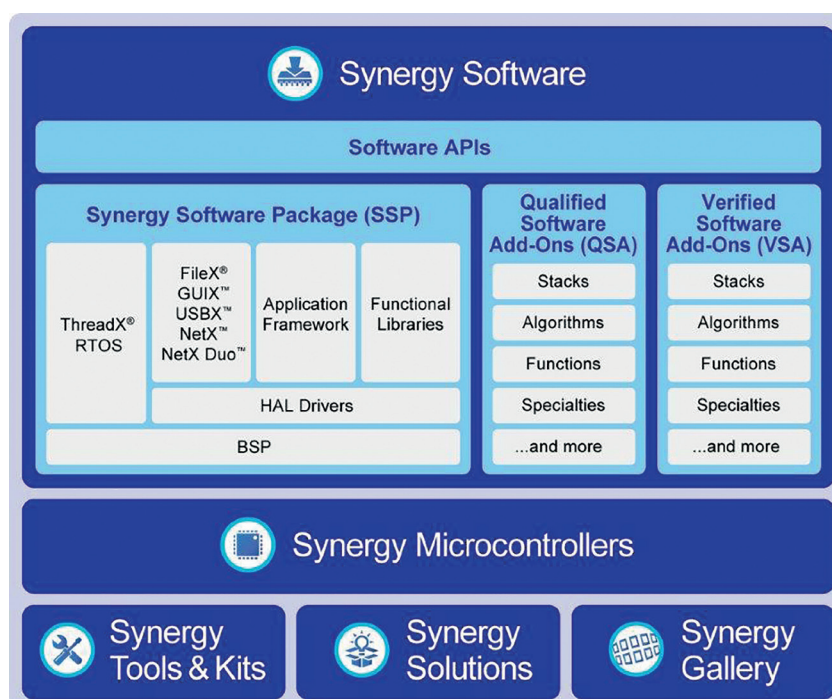
Figure 1-1: The five elements of the Synergy Platform

What makes Synergy Platform unique is that it is supported as a product by Renesas – warranted, maintained and supported. It is also a designer's dream because it makes their life very easy, enabling them to develop from the API and having the benefit of a single trustworthy point of support – Renesas. Purchasers are not forgotten either, as development can start without any up-front cost, plus the assurance of competitive device pricing when production starts. Just to repeat – no up-front cost – you go to the Synergy Gallery, download the Synergy Software Package (SSP) and the tool chain, and start development...   … IMMEDIATELY!

Now what follows is a more engineering oriented description of the platform which is worth reading if you're the kind of engineer that wants to know everything. But, if on the other hand, you can not wait to start your own development at the API layer, then feel free to skip the following chapters and go directly to either chapter 4 "Getting the Synergy Platform Toolchain Up and Running" (explains the installation of the toolchain), or to chapter 8 "Hello World – Hi Blinky" (describes a first application on the Synergy Platform).

The key parts of the platform have been optimized to work together, creating new blocks where necessary but also reusing proven IP where it is already cutting edge. For the developer this ensures that everything works together without a glitch and that software development can start directly without having to worry about underlying layers, initialization of peripherals or setup of the Real-Time Operating System (RTOS). This relieves developers from the burden of taking care about these basic tasks and helps reduce time to market.

The ARM Cortex-M based MCUs have been designed with software in mind and are extremely scalable in either way – both up and down – as registers maintain their addresses and bit-positions throughout the different series. There is a commonality of peripheral IP across the different series. This means that a move from one series of microcontroller to another will, in most cases, simply require a recompilation of the project with the new settings and a high-level of pin-compatibility allows for extremely easy PCB layout.

Last, but by no means least, the Synergy Gallery on the Web provides a very convenient, single point of access to everything needed for the development of a Synergy application: Compilers, development environments, tools, documentation and support, not only from Renesas, but also from third-parties. So with Synergy Platform, a complete ecosystem is available at your fingertips!

1.1 Challenges in Todays Embedded System Design

Embedded systems have changed over the last decades. Prior to the turn of the century they employed few simple interfaces such as push-buttons for input or a character LCD for output and used a single function, mostly implemented as simple loop inside `main()` with interrupts to handle a limited amount of tasks. A microcontroller with a few MIPS (million instructions per second), a couple of kB of memory and a basic serial communication would have been sufficient for this kind of system.

Today's embedded systems however are highly connected, having a wide variety of interfaces like Ethernet, wireless, or graphical user interfaces, all of which need to be configured / handled, and which exchange data and messages with each other and with the outside world to form the complete application. This can require an MCU with a clock speed of 100 MHz or more, several MB of flash memory and perhaps 128 kB of RAM. A Real-Time Operating System (RTOS) can be helpful, if not essential, as different threads need to be prioritized and executed concurrently. Development of such systems is no longer possible in the way that legacy systems were designed, as increased connectivity requirements make these systems less hardware and more software centric.

In addition development cycles get shorter and new feature requests come in more often. All of this not only places a high burden on the developer who has to tackle new challenges more frequently, but is also a huge investment, not all of which may be visible right from the beginning.

A pre-integrated platform offers a low barrier to entry and minimizes the total cost of ownership, as everything is visible upfront, helping to speed up the development, reducing time to market and keeping the projects budget on track. All this allows more features to be integrated in the available time and at the same cost and therefore to create heavily differentiated products.

It is also worth mentioning that Synergy Platform requires no upfront expenditure. Purchase just one Synergy MCU and you can access commercial software with a value of several hundred K$, with no limit on development seats, number of end-products, or number of total MCUs.

1.2 Synergy Software Package

At the heart of the complete Synergy Platform ecosystem is the Synergy Software Package (SSP), specifically optimized for the Synergy MCU architecture. It integrates Application Frameworks for system level services, Functional Libraries containing specialized software, for example for digital signal processing, very efficient Hardware Abstraction Layer (HAL) drivers for all peripherals, and the Board Support Package (BSP) including start-up code for boards and MCUs.

In addition to the software mentioned above the SSP also includes renowned and proven packages from Express Logic such as the ThreadX® RTOS and the X-Ware™ communication stacks and graphics middleware. The SSP covers all basic functionalities like initialization of the microcontroller and its peripherals, or the setup of the RTOS. Many engineers spend months, if not years, developing this kind of low-level software rather than being able to focus their time on differentiating in the application – this is the power of Synergy Platform!

If you are curious about the details of the SSP please be patient. We will cover them in chapters 2 and 3.

1.3 The Synergy Microcontroller Family

Looking at the family of Synergy MCUs there are initially four series – S1, S3, S5 and S7 – for use in end-products ranging from small, battery operated sensor applications, to high-performance, processing-intensive embedded systems. Built-in peripherals for connectivity, graphic displays, security and more make them well suited for the rapidly expanding Internet of Things (IoT) market, but by no means limited to that.

All Synergy MCUs are based on 32-bit ARM Cortex-M cores (see Figure 1-2 for the initial series available): The S1 Series are based on the M0+ core, while the S3, S5 and the S7 Series are based on the M4 core. All of them include standard peripherals from ARM like the Nested Vectored Interrupt Controller (NVIC) or the ARM Memory Protection Unit. Moreover Renesas added its own IP where ARM has no solution and where additional performance or features where needed. This additional IP is based on proven technology from Renesas, adapted to the compatibility and scalability demands of the Synergy Platform.

Figure 1-2: The initial series of the Synergy MCUs

The initially released MCU series exhibit the features below.

- **S1 Series – Ultra-Low Power:** Core frequency up to 32 MHz. Up to 256 kB Flash memory. Ultra-low active power of 70.7 µA per MHz. Voltage range from 1.6 V to 5.5 V. Peripherals include data converters, timers, capacitive touch sensing unit, serial interfaces, safety and security.

- **S3 Series – High Efficiency:** Includes an FPU (floating point unit). Core frequency between 32 MHz and 100 MHz. Up to 1 MB Flash memory. Voltage range from 1.6 V to 5.5 V. Peripherals include data converters, timers, segment LCD-controller, USB and serial interfaces, safety and security.

- **S5 Series – High Integration:** Includes an FPU. Core frequency between 100 MHz and 200 MHz. Up to 2 MB of Flash memory. Voltage range from 2.7 V to 3.6 V. Peripherals include data converters, timers, graphics LCD-controller, capacitive touch sensing unit, Ethernet MACs, USB, CAN, serial interfaces, safety and security.

- **S7 Series – High Performance:** Includes an FPU. Core frequency between 200 MHz and 300 MHz. Up to 4 MB of Flash memory. Voltage range from 2.7 V to 3.6 V. Peripherals include data converters, timers, graphics LCD-controller and a 2D drawing engine, Ethernet MACs, USB, CAN, serial interfaces, safety and security.

Each series will expand gradually as new devices are introduced and Renesas is already planning additional series with yet more innovation on board!

All of the MCUs in each series are feature and pin compatible. This allows easy scalability and code reuse from one device to another. Developers benefit from that as it is not necessary to choose the final device at the very beginning because changing to a different one later on is easy. Compatibility is not only maintained across each series but mostly between all four series of the platform as well. This not only holds true for the peripherals but also for the pinout. For example, the members of the S3 Series and the members of the S7 Series in the LQFP-100 package feature the same pinout and are therefore drop-in replacements for each other. Similar packages over different Synergy MCU Series have pinouts which are extremely similar to each other. This way it is even possible to create circuit board layouts with multiple package footprints within each other for flexible manufacturing options of the end-product.

Figure 1-3 shows the key features and peripherals of the S7 Series of Synergy MCUs, representing the superset devices of the Synergy MCU Family, having the largest on-chip memory and the most complete set of peripherals. Our examples and projects throughout this book are based the Synergy S7 Series Starter Kit for that reason.
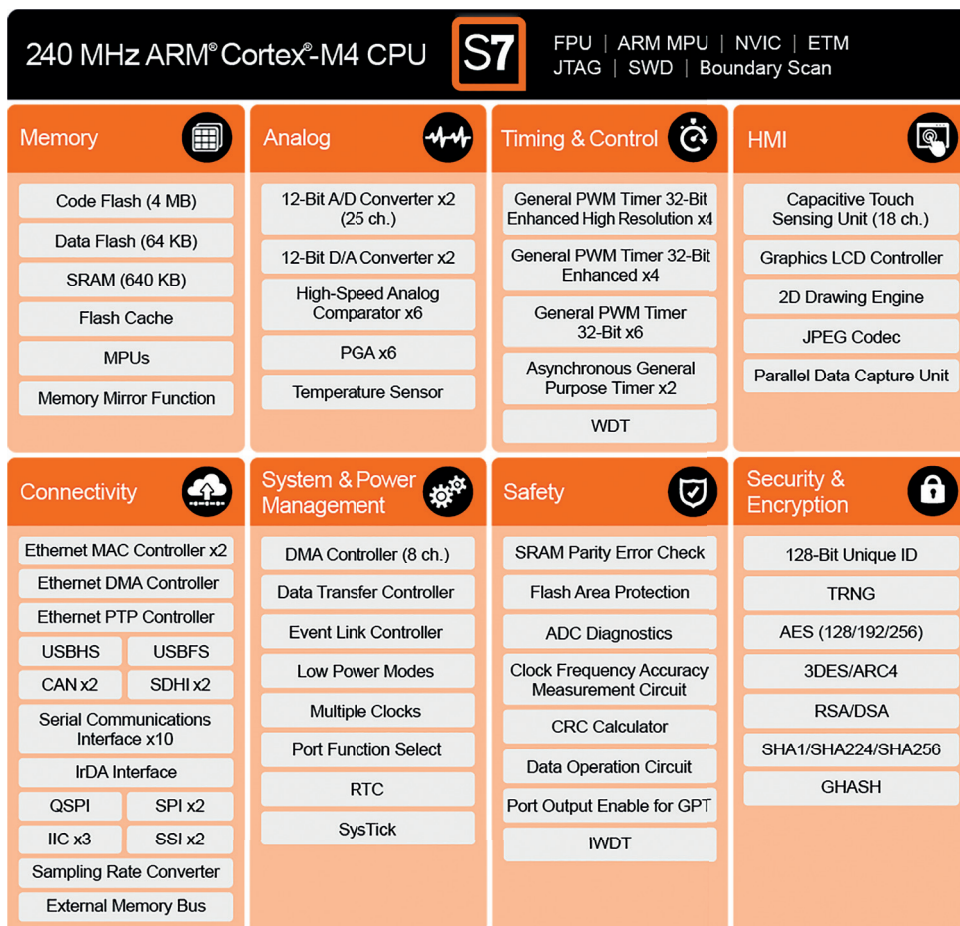
Figure 1-3: Block diagram of the S7 Series of Synergy MCUs

The peripherals in the smaller devices are mostly subsets of the ones found in the larger devices. For example, the RTC, the real-time controller, found in the S1 Series is a pure orthogonal subset of the RTC found in the S7 Series Synergy MCU. It lacks the time capture and tamper detect input and has a reduced alarm functionality, but the control registers have no dependencies as they are scaled down to the lower feature set. Additionally, the offsets of the control registers are constant, even as features are removed. This means that

software written for the RTC of a S1 Series device will work on a S7 Series device without modification. Scaling down from a S7 to a S1 Series device just means that only those lines of code need to be changed which belong to a feature not available on the smaller device, making the life of developers a lot easier. This clearly shows that there was a lot of thought put into the silicon.

1.4   The Renesas Synergy™ Tools and Kits

Renesas has taken great steps to create useful software and hardware development tools that can be used to explore the platform's technical capabilities and that will take the user beyond the evaluation stage, all the way to production.

The available software development tools include the following:

- **e² studio:** The Eclipse-based Integrated Solution Development Environment (ISDE) from Renesas including special configurators for the Synergy Platform. Uses the GCC ARM® Embedded code generation tools.

- **IAR Embedded Workbench® for Renesas Synergy™:** The Integrated Development Environment from IAR Systems® for the Synergy Platform, using IAR's proprietary ARM® Cortex®-M C-compiler. Works with the Renesas Synergy™ Standalone Configurator (SSC).

- **TraceX®:** The RTOS event and behaviour visualizer and analysis tool from Express Logic.

- **GUIX Studio™:** A PC based program from Express Logic for designing graphical user interfaces.

e² studio contains all the tools necessary to create, compile and debug projects for the Synergy Platform. It is called ISDE as additional solution-oriented components and plug-ins have been added, making it more powerful. This is especially true for the configurators, which allow an easy graphical access to the different hardware features like the clock module or the pin configuration without the need for deep study of the user's guide. These configurators will create all the necessary settings and the initialization code automatically and include an error checking feature to detect problematic settings already at design time.

During development, hardware will be needed to run first tests. In most cases, this will be necessary well before the user's own board is ready. For that purpose, Synergy Platform offers two different types of kits, and both come with an on-board J-Link® debugging and programming interface:

- **Synergy Starter Kit (SK):** For general evaluation. Uses an S7G2 Group Microcontroller (MCU), as this is the superset device, giving a good introduction to the Synergy Platform. You can evaluate the complete ecosystem with this kit and later on pick any device from the different families for your own project, as everything is also applicable to the smaller siblings. About 80% of the pins of the MCU are accessible through connectors, so attaching your own hardware is not a problem, an important capability feature for evaluation.

- **Synergy Development Kits (DK):** For full project prototyping with access to all pins through many expansion ports and up to four Pmod™ connectors. They feature a Bluetooth® low energy radio for wireless connection to a mobile device and many other specialized expansion boards.

1.5   The Synergy Solutions

For special needs, like human-machine interfaces (HMI), a smart sensor network or industrial networking, the Synergy Solutions provide specialized kits, which go beyond the typical embedded product development hardware. There are two different flavours of them:

- **Synergy Product Example (PE):** Represent how an actual end product would be designed. They come with the necessary hardware, software and a complete documentation, including schematics, layouts, BOM, Gerber files and a special "design journey documents", describing the methodology of how and why design choices were made and the reasoning behind the selection of components during the design process.

- **Synergy Application Example (AE):** Technology building-block examples that can be used to build an application upon. They include multiple components creating a showcase how multiple technologies can be used to build a product. For example, the AE of a cloud connected system would demonstrate the use of different wireless networks, of a Human Machine Interface (HMI) and of cloud connectivity and services.
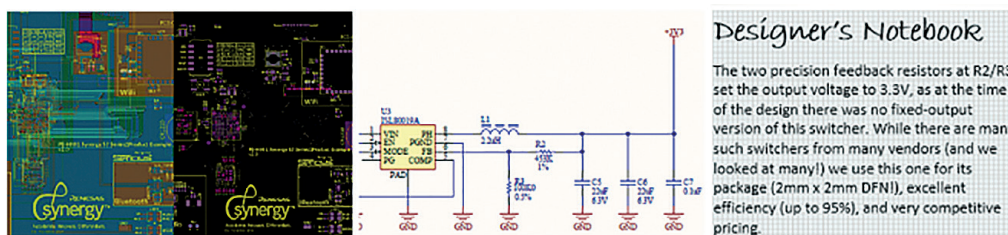
Figure 1-4: Product examples include schematics, PCB layout files, Designer's Notebook and much more

Synergy Solutions build on the Synergy Software Package, as well as on VSAs (Verified Software Add-Ons) and QSA (Qualified Software Add-Ons) and PPs (Partner Projects) from third-parties. These augment the software options available in the SSP providing yet more development convenience and evaluation versions of them can be downloaded from the Synergy Gallery. More details on VSAs, QSAs and PPs are provided in chapter 12.

1.6   The Synergy Gallery

The Synergy Gallery is your single point of entry for everything related to Synergy Software, Synergy Tools and licensing. Plus software and services from third-party vendors who are participating in the Synergy Platform ecosystem. It requires a one-time registration, after which access to all the software and documentation needed for a development is granted. It is also the place to request the development / production license for use in an actual development and for mass production.
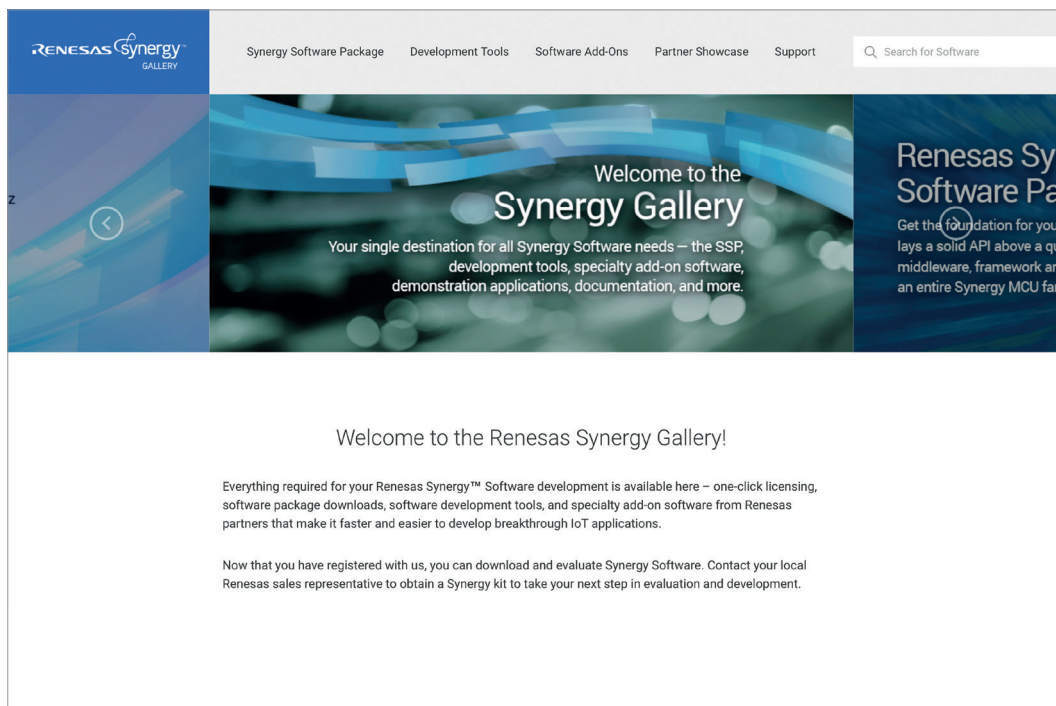
Figure 1-5: The welcome screen of the Synergy Gallery is the gateway for the access to all tools, software and support

And if things go wrong during your development, help is available from the Renesas Gallery. Just select *Support* on the top of the page which will get you to the support homepage, where you can choose from a variety of options. A good place to start it the Knowledge Base (**https://en-eu.knowledgebase.renesas.com/**) or the Renesas Rulz Forum (**http://renesasrulz.com/synergy/**). Here you can ask your question and will get a response 24 – 5 from a Renesas support engineer.

In the future, the Synergy Platform will include a complete and secure cloud accessible infrastructure for your end products as well.

Points to take away from this chapter:

- The Renesas Synergy™ Platform consists of five elements:
    - Synergy Software
    - Synergy Microcontrollers
    - Synergy Gallery
    - Synergy Tools and Kits
    - Synergy Solutions

2 DETAILS OF THE RENESAS SYNERGY™ SOFTWARE

What you will learn in this chapter:

- What the different components of the Synergy Software Package (SSP) are and how they are layered.
- Details of the different layers and how they work together.
- Specifics of the X-Ware™ middleware and the ThreadX® RTOS.

The best microprocessor cannot unlock its full potential if either the software running on it is not up to the task, or the software ecosystem is too complicated to be useful to the developer. A well thought-out concept like Synergy Platform helps to create applications which are easy to develop, simple to maintain, and which make the best out of the performance and the features of the microcontroller (MCU).

The simplicity found in the Synergy Platform was attained by making all major parts, MCUs, software, tools and kits and application solutions work perfectly together.

The Synergy Software Package (SSP) was specifically optimized for the Synergy MCU architecture, which, in turn, was developed with the software in mind. The SSP integrates application frameworks, functional libraries, the hardware abstraction layer (HAL) drivers, and, as the basis for all of that, the Board Support Package (BSP), together with the widely used ThreadX® Real-Time Operating System (RTOS) and the X-Ware™ middleware from Express Logic, which are pre-licensed for unlimited development and production usage if used together with the SSP. This results in a single and easy to use software package where all the functionality can be accessed by a simple and robust API (Application Programming Interface).

The complete SSP, even the parts created by Express Logic, are fully supported by Renesas. This means that you, as a developer, will have only one point of contact in case you need support. No dealing with different support channels, for everything inside the Synergy Platform there is just one: Renesas! Having worked a lot with different software packages from different vendors in my projects, I know what this means: Life is much easier this way, as there is no need to identify and talk to different people about the issue at hand, and eventually playing the interface between them!

Within any Synergy Software development environment, developers normally have complete on-screen visibility of all SSP source code during development and debug. However, some SSP software modules are protected, meaning that while they can be viewed on-screen and while the source will be used for compilation to achieve a small footprint, they cannot be altered, printed or saved to a file outside the development environment (see Figure 2-1). In the event that you need to go further, an SSP Source Code License available from Renesas enables the transformation of protected software modules into clear text files that can be edited, printed, and saved to a file. SSP Source Code Licenses can be purchased for individual or all SSP software modules.
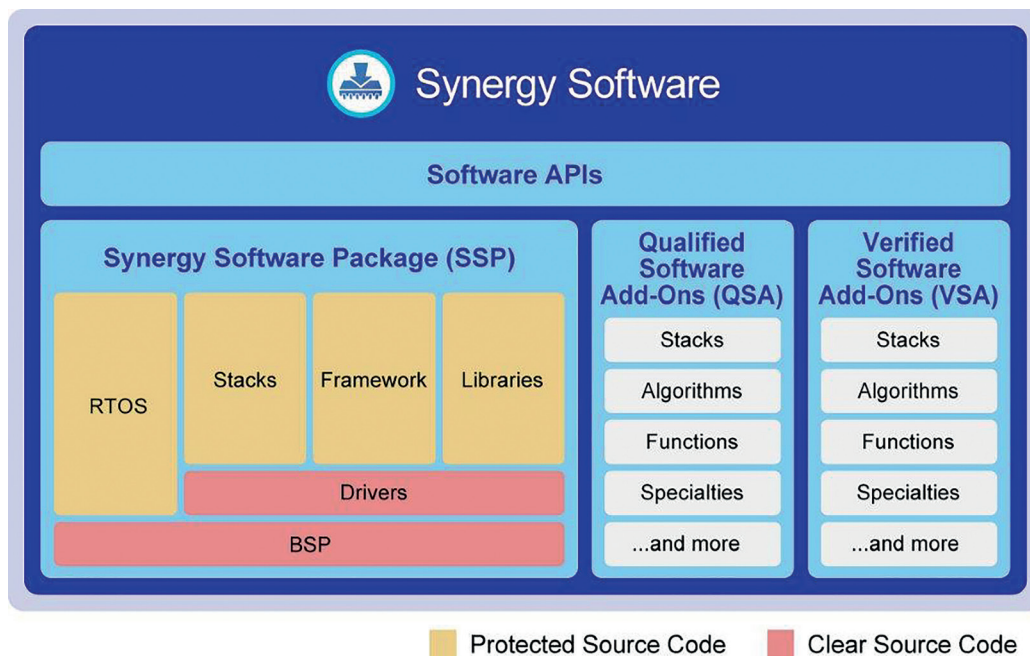
Figure 2-1: Parts of the source code is protected from editing, but can be unlocked by obtaining a source code license

2.1 Introduction to The Synergy Software Package (SSP)

As mentioned before, the Synergy Software Package (SSP) is a comprehensive piece of software covering all aspects of an embedded systems software development. It comprises the following parts:

- The **Board Support Package (BSP)**, customized for every Synergy hardware kit and Microcontroller. It includes the startup-code for all supported blocks. Developers using custom hardware can take advantage of the BSP, as it can be tailored for end products and your own board by using the Custom Board Support Package Creator, which can be downloaded from the Synergy Gallery.

- The RTOS-independent **HAL-drivers**, providing very efficient drivers for all peripherals and systems services. They eliminate a lot of deep study of the underlying hardware in the microcontroller as they abstract the bit-settings and register addresses from you.

- The **Application Frameworks**, containing the system level services linking the RTOS to the Hardware Abstraction Layer (HAL) for interprocess messaging, security services, serial communication, audio playback, capacitive touch sensing, Bluetooth® low energy and much more. The completeness of these frameworks reduces errors and saves time during the development of an application.

- The **Functional Libraries** containing, for example, specialized software for digital signal processing or security and encryption related functions also reduce development time and improve the stability of the end-application.

- The **ThreadX® Real-Time Operating System**, provides a multitasking real-time kernel with preemptive scheduling and a small memory footprint. ThreadX® has been deployed in over 5.5 billion embedded devices in various areas.

- The **X-Ware™ stacks and middleware**, including file systems (FileX®), graphical user interfaces (GUIX™), USB and TCP/IP communication (USBX™, NetX™ and NetX Duo™). Everything here is completely optimized for, and integrated into, the Synergy Platform.
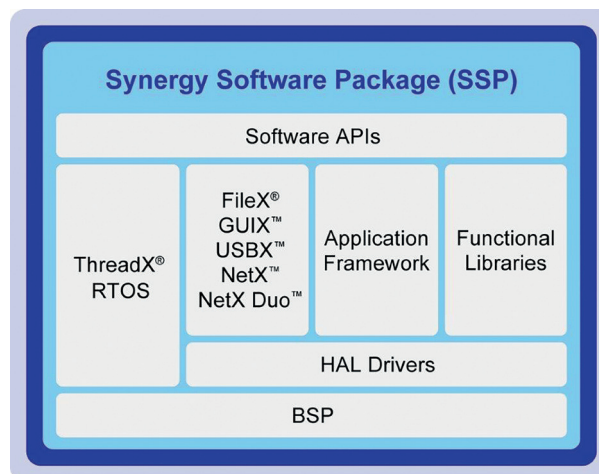
Figure 2-2: Everything in the Synergy software ecosystem is built in layers

Figure 2-2 shows the architecture of the different layers. Each of them can be accessed by API calls directly, or in a stacked manner. An audio application, for example, can call the audio section of the Applications Frameworks, which then uses the I^2S part of the HAL to provide read / write access to an external audio converter using the serial port. Of course, an end application could also access the HAL or the BSP using API calls directly, but going through the frameworks is much more convenient.

All SSP modules support applications written in C++ and C++ applications can be configured and used in e^2 studio and IAR Embedded Workbench® for Renesas Synergy™.

The SSP is "production grade" software, developed using industry best practices and compliant with MISRA C:2012, the ISO/IEC/IEEE 12207 software life cycle process and the CERT® 2nd edition. It comes with a comprehensive software datasheet stating benchmarks, code size, execution times, and the testing conditions under which these results were obtained. Renesas provides more details about their extensive quality assurance for Synergy Platform on the Synergy Gallery.

Updates and bug-fixes to the SSP will happen regularly, with the scheduled releases following a roadmap and each major or minor release introducing new features. For the long term, Renesas has a great reputation for supporting their products way beyond the point at which other manufacturers would cease maintenance. With the SSP, bug fixes and enhancements are also applied to the prior major release, with bug fixes continuing to be implemented in all releases until the production end of the microcontroller, plus one year. It is lifetime maintenance, ensuring that software developers can continue working with their current baseline without having to worry about being forced to switch to a new release right away due to bugs not fixed in their version. Personally, having supported customers working on large (and long) projects for quite a while, I consider that as a great feature, as you do not need to change toolchain-versions during development. You are not left alone with software distributed on an "as is" basis.

2.2   Introduction to the Board Support Package (BSP)

The Board Support Package (BSP) on the bottom layer of the software architecture is a requirement for any SSP project and its responsibility is getting the MCU from reset to `main()`. Its code will set up the stacks, clocks, interrupts and the C runtime environment, before reaching the user's application. It also configures and sets up the port I/O-pins and performs any board specific initialization.

Therefore, this package is specific to a combination of a board and an MCU, which is selected during design in e^2 studio using the different configurators of the ISDE (Integrated Synergy Development Environment). Every development board provided is supported

by a BSP. The configurators inside e² studio will extract the necessary files from the SSP and configure them based on the settings made in the user interface. The Board Support Package is heavily data driven and consists of configuration files, header files and an API.

The core of the BSP itself is compliant with the CMSIS (ARM® Cortex® Microcontroller Software Interface Standard), following the requirements and the naming conventions of that standard.

The BSP provides public functions, available to any project using the package, that allow access to the functionality that is common across the MCUs and boards supported by it. Functions include locking/unlocking the hardware and software, interrupt handling, like registering callback functions and clearing flags, software delay and register protection. The names of these functions start with `R_BSP_`, and associated macros with BSP_ for an easy differentiation from other parts of the SSP. The only exception are functions providing functionality described in the CMSIS-Core. Additionally, the BSP includes a function that returns information (number and I/O-pins) about the LEDs used on a board inside a structure.

New boards and devices will be added to the BSP once they are available, assuring a solid long-term base for current and new designs. Board level support for custom boards can easily be generated using the Custom BSP Creator, which can be downloaded from the SSP-page of the Synergy Gallery, together with an application note explaining the procedure.

2.3   Introduction to the HAL Drivers

On top of the BSP sits the Hardware Abstraction Layer (HAL), which provides device drivers for the peripherals and which aligns with the registers of the MCU to implement easy to use interfaces, insulating the programmer from the hardware. It is a collection of modules and each of them is a driver for a peripheral available in a Synergy Microcontroller like the SPI (Serial Peripheral Interface) or the ADC (Analog-to-Digital Converter), and their names begin with $r\_$ for an easy differentiation from other parts of the SSP. These modules are inherently RTOS independent and are composed of two components: A low-level driver (LLD), which manipulates the registers of a peripheral directly and uses different versions of the same peripheral seamlessly and a high-level driver (HLD), whose code is specific to a Renesas hardware peripheral, but which does not access the registers directly. The HLD exposes the application programming interface to the frameworks or the user program and makes use of the LLD to interface with the microcontroller. The benefit of this architecture is that the LLD allows for very fast code and that HLD makes the APIs portable across the different Synergy MCU Series.

The interface to abstract the hardware is consistent throughout all modules of the HAL and can be extended. Some of the peripherals support multiple interfaces, while some interfaces are supported by multiple peripherals. The advantage of that is the flexibility gained, as requirements can be modified at a higher level. If, for example, the code was originally written for the dedicated hardware IIC peripheral, but later on the IIC-functionality of the SCI (Serial Communication Interface) should be used, it is sufficient to change the configuration information. The application code itself remains unchanged. While the API functionality can be accessed directly through the HAL interfaces, most of the functions can also be accessed through the different frameworks available in the SSP.

MCU Reset

ResetHandler()

MSP is set automatically from value at adress 0 x 00000000

Calls Systeminit

Jump to main()

User Application

[1] If supported by MCU and enabled in compiler

[2] These operations will vary depending on settings in bsp_cfg.h

[3] These operations will vary depending on settings in bsp_pin_cfg.h

Systeminit()

Initialize FPU [1]

Warm Start – Pre C runtime initialization hook

Configure group interrupts (system exceptions)

Configure System Clocks Call CGC Module [2]

- Set clock dividers and multipliers
- Stop unused clocks
- Wait for clock stabilization
- Transition to selected clock

Set CMSIS system clock variable

Initialize Pins Call IO Port Module [3]

Initialize C runtime data

- Zero out uninitialized data areas
- Copy initialized RAM data from ROM to RAM

Warm Start – Post C runtime initialization hook

Initialize ELC events for NVIC triggers

Initialize register protection bsp_register_protect_open()
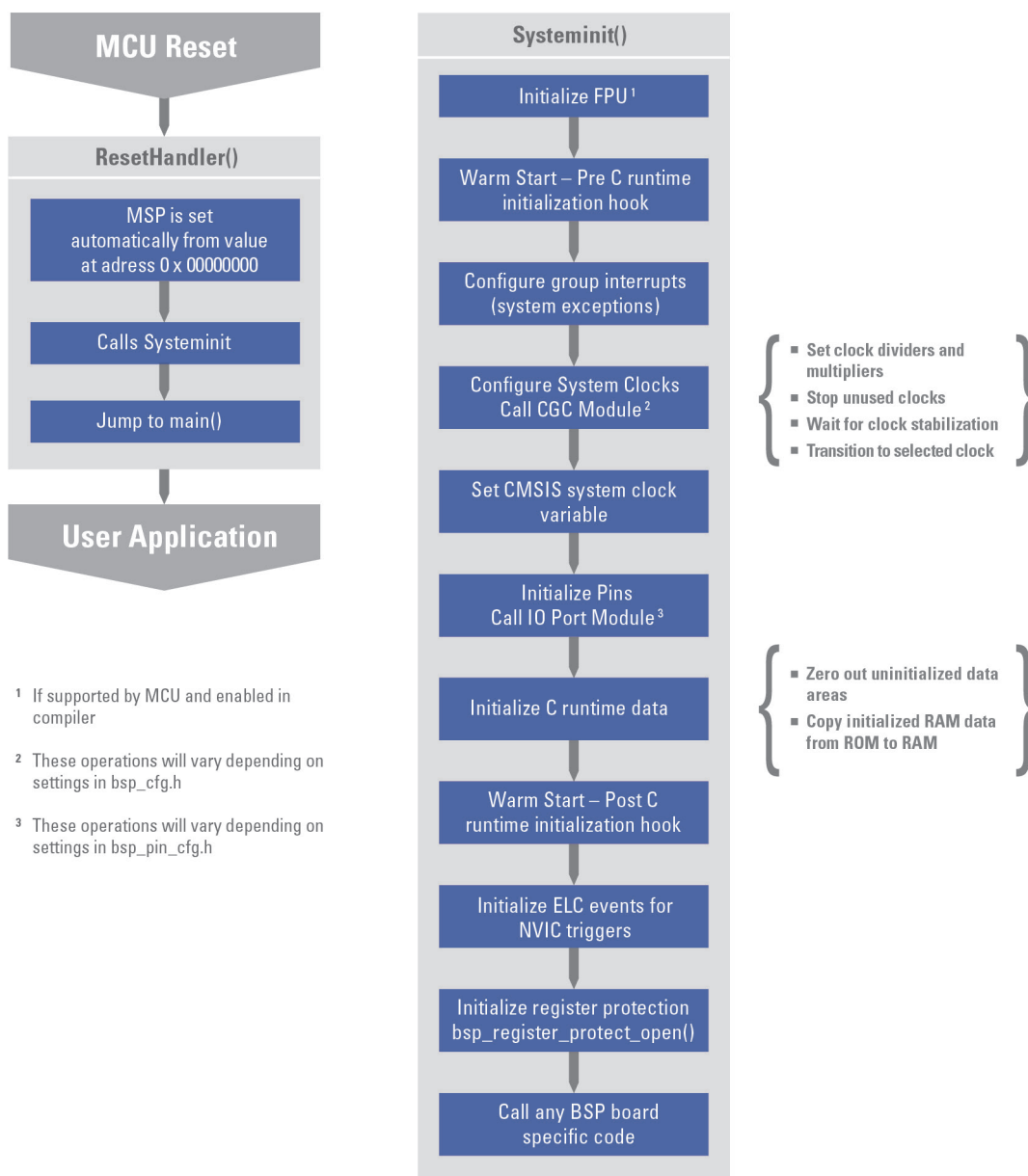
Call any BSP board specific code

Figure 2-3: The flowchart for the BSP showing the move from reset to main()

2.4 Introduction to the Application Frameworks

Sitting on top of the HAL, the Application Frameworks provide developers with an even higher level of abstraction, allowing for better reuse of code and ease of programming, resulting in a shorter development time. The frameworks provide abstraction of various system-level and technology-specific services, enabling rich functionality with simple APIs. The different frameworks are integrated with the ThreadX® RTOS features to manage resource conflicts and synchronization between multiple user threads.

Application Frameworks are available, for example, for peripherals like SPI, ADC or IIC, and for common services like audio, cap-touch sensing or JPEGs. Many of the frameworks use the services of each other, and also combine several HAL and BSP calls. For example, for timing functions, the ADC framework uses services from the GPT timer interface and from the DMA or DTC for the efficient transfer of data through the shared interface.
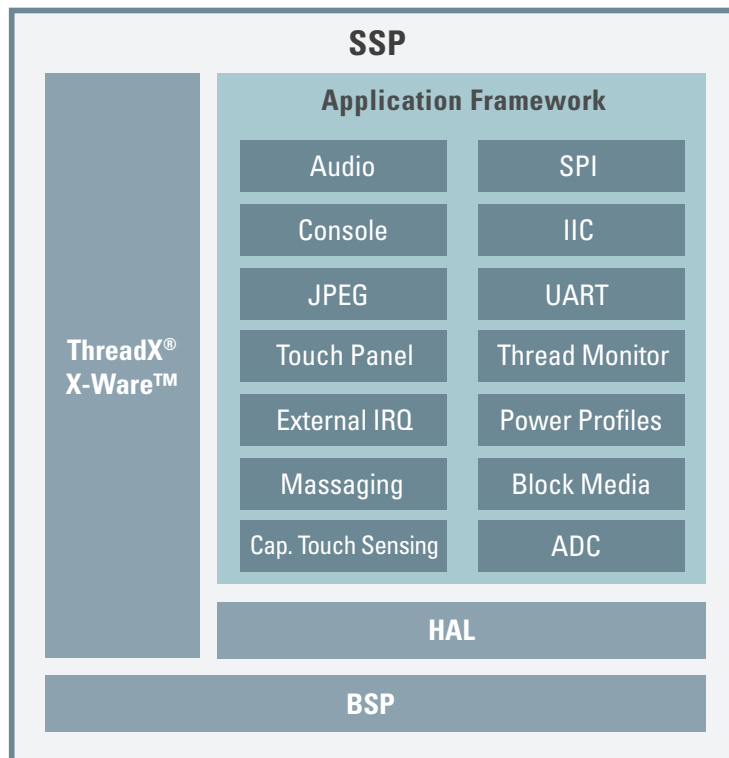

Figure 2-4: The Synergy Application Frameworks provide abstraction to various system-level and technology specific services

Developers using the application frameworks benefit from a higher level of abstraction, re-use of code across processors, and therefore potentially products, more consistent code, and are offloaded from the task of having to reinvent commonly used tasks across applications. At the same time, the Synergy Application Frameworks reduce the need to directly interface with the hardware available in the microcontroller even further. All module names begin with `sf_` for easy differentiation from other parts of the SSP.

2.5   Introduction to the Functional Libraries

Functional Libraries add more capabilities to the Synergy Platform and they interface directly with the HAL. They include extremely useful and carefully optimized functions and are callable through easy to use APIs. Included are, for example, a hardware accelerated Digital Signal Processing (DSP) library using the ARM® Cortex® Microcontroller Software Interface Standard (CMSIS), including over 60 functions for various data types, and a hardware accelerated cryptography and security library with a rich set of cryptography algorithms.

The Functional Libraries come from Renesas and other vendors, and are pre-tested, qualified, verified and supported by Renesas, assuring the highest possible quality of the code.
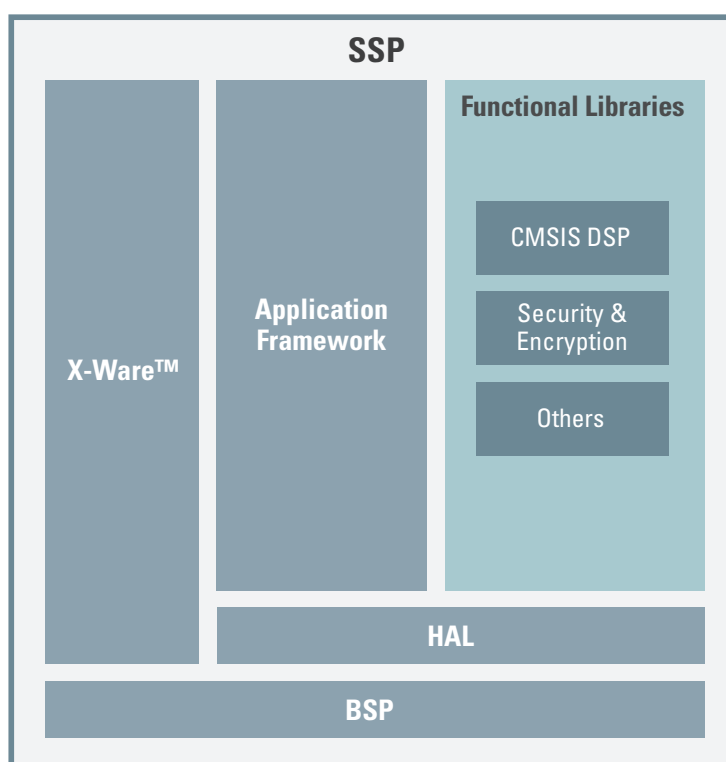
Figure 2-5: The Synergy Functional Libraries add additional capabilities to the Synergy Platform

Additional libraries will be added in new versions of the Synergy Software Package (SSP) over time, integrating even more features and functions, reducing the learning curve for new microcontroller developers and cutting the time-to-market for applications even further.

2.6 Included Middleware from Express Logic (X-Ware™)

In addition to all the software provided by Renesas, the SSP also includes additional middleware-libraries created by Express Logic and supported by Renesas for file-handling (FileX®), Graphical User Interfaces (GUIX™), USB communication in host and device mode (USBX™) and network communication (NetX™ and NetX Duo™). These libraries, as well as the ThreadX® Real-Time Operating System, have been developed with a consistent API structure and coding style, always keeping the ease of use for the developer in mind. This unparalleled consistency make the Synergy Software Package well equipped to speed development of new products for fast time to market.

2.6.1 FileX®

Express Logic FileX® is a high-performance FAT (File Allocation Table) file system for the SSP, developed especially for fast execution, fast-seek logic and small footprint. It is fully integrated with the SSP and ThreadX® and is available for all supported Synergy MCUs.

Supported are all FAT formats, including FAT12, FAT16, FAT32 and exFAT (extended FAT). FileX® is built especially for real-time embedded systems, maintaining an internal FAT entry cache. Other features are:

- Contiguous file allocation
- Consecutive sector and cluster read / write
- Long filenames and 8.3 naming conventions
- Unlimited creation of FileX® objects like media, directories and files
- Optional fault tolerance

FileX® supports USB mass storage devices, SD / eMMC cards, SPI flash, QSPI flash and on-chip flash, as well as RAM as storage media.

A high reliability is achieved through error detection and recovery capabilities, different fault tolerance options and built-in performance statistics.

The library interfaces to the other parts of the SSP through the Block Media Interface (`fx_io`), an abstract interface using function pointers instead of direct function calls, acting as adaption layer for the block media device drivers. Functions are called between file systems and the Synergy Platform block media drivers in the Application Frameworks, such as SD/MMC and SPI Flash, which in turn depend on the SDMMC, Flash and SPI interfaces of the HAL.

The interface remains the same for any media driver. All media drivers appear functionally identical at the file I/O layer and can be interchanged with one another without changing application code.

2.6.2   GUIX™

Express Logic GUIX™ runtime library is the SSP's high-performance Graphical User Interface framework that is optimized for the Synergy MCU architecture and enables the creation of elegant user interfaces. It is fully integrated with the ThreadX® RTOS and was designed to have a small footprint. It is implemented as a pure C library and only brings features used by the application into the final image. Due to a minimal function call layering and optimized clipping, drawing, event handling, and support for the Renesas hardware graphics acceleration, the response is quite fast.

The runtime library of GUIX™ supports multiple screens and multiple languages with UTF-8 string encoding. It includes functionality for horizontal, vertical and drop down lists, single- or multi-line text for either view or input, checkboxes, buttons, sliders, and scrollbars. The supporting framework enables event queues and signals, windowing and viewport management, as well as clipping, deferred drawing and dirty list maintenance.

GUIX™ has a dedicated desktop design application called GUIX Studio™, which provides a complete WYSIWYG screen design environment, where developers can drag-and-drop graphical elements to build compelling user interfaces, which can be executed on a Windows® workstation within GUIX Studio™, allowing quick and easy demonstration and evaluation of user interface concepts.
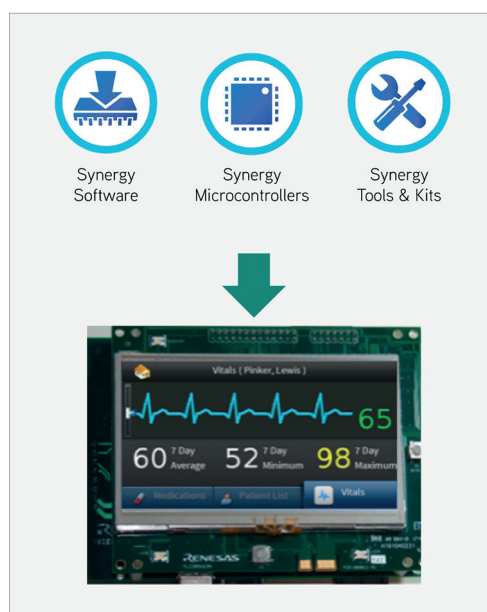

Figure 2-6: GUIX™ allows the easy creation of compelling GUIs

Once the UI is finalized, GUIX Studio™ will generate C code compatible with the GUIX™ library, ready to be compiled. Developers can produce pre-rendered fonts for use within an application, and fonts can be generated in monochrome or anti-aliased formats using the integrated font generation tool that supports any set of characters, including Unicode characters for multi-lingual applications. GUIX Studio™ allows customization of the default colors and drawing styles of GUIX™ widgets, allowing developers to tune the appearance of their application to their needs.

As always, functionality of the runtime library gets called through intuitive, readable and highly functional API calls. Building applications based on GUIX™ is easy: include the `gx_api.h` file in the application source code and link the objects with the GUIX™ library.

2.6.3   USBX™

UBSX™ is Express Logic's small footprint and high-performance USB host and device stack for SSP applications, allowing a Synergy MCU to interface with USB devices or to be connected with Windows®, Apple® and Linux workstations over USB. It is fully integrated with ThreadX® and available for all supported Synergy MCUs.

The stack includes support for the Low Speed (1.5 Mbps), Full Speed (12 Mbps) and High Speed (480 Mbps) transfer modes and the USB 1.1 and USB 2.0 specifications. It can act in two modes: host and device.

The host mode of USBX™ is used when the application requires communication with certain USB devices such as a USB keyboard, a USB printer or USB Flash disk, this means with peripherals acting as slaves. There are two major components responsible for this mode: The USB core stack and the USB controller.

The USBX™ core stack is responsible for detection of device insertion and removal, as well as all the protocols available to USB (control, bulk, interrupt and isochronous). The core stack ensures correct device detection and configuration, and its plug-and-play mechanism searches for a USB class driver responsible for this device.

The USB controller supports major USB standards like OHCI and EHCI. It is possible for a single host system to have multiple host controllers. The USB class driver(s) are responsible for driving each USB device once it has been enumerated. Several USB standard classes are supported, such as CDC, HID, HUB, STORAGE, AUDIO or PRINTER, but proprietary classes can be used as well. USBX™ host mode can support cascading hubs, multiple configuration devices, and composite devices.

The device mode of USBX™ is used when the application should communicate with a Windows®, Apple® or Linux desktop. In this case the Synergy MCU is considered to be a USB device or slave. The architecture for the device side is similar to the host side although simplified. The USBX™ device core stack is responsible for handling the USB host enumeration requests. The USB device driver class is responsible for driving the USB device once enumeration took place by the host. In the device mode, USBX™ can support a complex device framework including multiple configurations, multiple interfaces, and multiple alternate settings. As with the USB host mode, support for several of the standard USB classes like CDC, HID, STORAGE, PIMA or RNDIS is included in the library.

Working with the USBX™ library is very straightforward and a first working version can be accomplished in a very short timeframe. All what needs to be done is to add and configure a thread, a queue and four modules in the Synergy configurator in e² studio. In the exercise provided in chapter 10, you will experience this simplicity yourself.

2.6.4   NetX™ / NetX Duo™

Express Logic's NetX™ and NetX Duo™ implement a TCP/IP protocol application for the SSP. While NetX™ provides a streamlined IPv4 capable TCP/IP stack, NetX Duo™ provides IPv4 and IPv6 capabilities in a dual-stack product, offering a future proof solution.

NetX™ and NetX Duo™ have a unique Piconet™ architecture, where only those services and protocols that are actually used are included in the image. Combined with a zero-copy API, which eliminates the processor cycles normally consumed by moving data to and from user buffers, they are perfect for applications that require network connectivity, as applications can use the freed processor cycles for more useful tasks. NetX Duo™ conforms to RFC standards, verified by the industry standard Ixia IxANVL™ test suite, and has achieved IPv6-Ready Logo certification, evidence that it has passed conformance and interoperability tests, administered and validated by the IPv6 Forum.

In addition to the zero-copy API, NetX™ also provides a BSD socket-compatible API for applications with legacy BSD application code. The application may create any number of packet pools in any number of memory areas. These zero-copy packets can be linked with packets from the same pool or even a different pool to accommodate larger payloads.

Basic UDP packets pass through NetX™ and NetX Duo™ without any copying or system context switches; they are delivered directly to waiting threads. This is referred to as UDP Fast Path™ Technology.

NetX™ and NetX Duo™ include network configuration protocols like DHCP (client / server) and SNTP (network time protocol), domain name services like DNS, mDNS, DNS-SD and NAT and email transfer with POP3 and SMTP. HTTP web server / network management is as well available, as are the connectivity protocols PPP, Telnet, FTP and TFTP. NetX Duo™ has received safety certifications according to IEC 61508 SIL 4, IEC 62304 Class C, ISO 26262 ASIL D and EN 50128 SW-SIL4.

Additional components like HTTPv6, FTPv6, DNSv6, Telnetv6, and DHCPv6 (client only) are available for NetX Duo™. Higher level solutions such as SSL/SSH/TLC are available as Synergy Verified Software Add-on (VSA) components. Information about them can be found in the Synergy Gallery on the Web.

2.7   The RTOS of Choice: ThreadX®

ThreadX®, created by Express Logic, is the Real Time Operating System (RTOS) of choice for the Synergy MCU Family. It was developed specifically for both high-end and graphic rich applications, as well as for embedded systems with limited memory and special requirements in terms of determinism. It features a small Flash-memory footprint (less than 2 KB on an S3, S5 or S7 MCU), a small RAM requirement (minimum of < 1 KB for kernel RAM) and a short context switching time of 0.7 µs on a Synergy R7FS7G2 MCU.

As a multitasking RTOS, ThreadX® uses multiple advanced scheduling algorithms, provides real-time event trace, message passing and interrupt management as well as many other services and is fully deterministic. It has a proven reliability record with more than 5.5 billion deployments in the consumer, medical electronics and industrial automation markets, and meets numerous important safety and quality standards.

Other advanced features offered are for example its picokernel™ architecture, Preemption-Threshold™ scheduling, Event-chaining™ and a rich set of system services. It also supports the X-Ware™ middleware with the services required by the different components.

2.7.1 Why Use an RTOS?

Not so long ago, the life of a software-developer for an embedded system was comparably straightforward. Most systems consisted of a simple main loop, where a single background task was running in an endless loop. It executed tasks like reading inputs from I/O-pins (for example the state of a push-button), performing certain calculations like multiplying a measured voltage and a current to obtain a power consumption, updating discrete outputs and so on.

The few peripherals connected to the microcontroller issued an interrupt to notify the CPU that maybe a new conversion result from a data converter is be available and should be read, or that the RS-232 port finished transmitting the last word and is waiting for new data. These asynchronous events have been traditionally handled inside the interrupt service routines. And if no interrupt was happening, the software was idling in the background loop, waiting for one to occur.

With embedded systems growing more complex, having a higher connectivity to the outside world, or more user interfaces to serve and more tasks to execute, the setup mentioned above was is hard to maintain, and an operating system is becoming not only something preferable, but a necessity. Why? The response time of the background loop was hard to predict and definitely not deterministic, as it was affected by the decision tree inside the code and would even change once a modification to the software was made. Another disadvantage was that all tasks (or "threads") in the background-loop had the same priority, as the code was executed serially, which means that the reading of the push-button and the power-calculation in the example above would always having been executed one after another, even if the button was pressed multiple times. In other words, some of these events could have been missed, something which is a no-go in embedded computing. The system needs to be more responsive and more predictable; it needs to be real-time.

Having seen the clear need for an operating system, the next question is: Why not use an off-the-shelf OS like Windows® or Linux? They are part of our everyday life and seem to do their job quite well. There are a couple of reasons for not using them. First of all, they offer by far too many features, not needed in an embedded system, and they are not configurable enough. Secondly, they need more resources and memory space than what is available in a resource-restricted embedded application. And last but not least, their timing uncertainty is still too large, to fulfill real-time requirements.

All of this means that there are special requirements for an OS running on our embedded systems. The first and most important one is the predictability of the timing behavior. A RTOS needs to be deterministic, which means that the upper limits of the blocking times, like the time during which interrupts are disabled, need to be known and available to the developer.

The second requirement is that the RTOS has to manage the timing and scheduling. It has to be aware of the timing constraints and deadlines and it must provide a time service with a high resolution. And finally, the operating system should be fast, small and configurable.

Another major function provided by an RTOS is thread management, allowing the execution of quasi-parallel tasks on an MCU using threads (a thread can be understood as a lightweight process) by taking care of the state of the threads, the queuing of the processes, and allowing for preemptive threads and interrupt handling. Other services provided to the application are scheduling, thread synchronization, inter-thread communication, resource sharing, and a real-time clock as time reference.

While many software designers balk at using an RTOS, because they fear its complexity and learning curve, real-time operating systems offer plenty of benefits like the increased responsiveness to real-time events, the prioritization and easy addition of threads, the reduced development time once the first steps are mastered, and services added to the application.

As with all good things in life, there are of course some disadvantages in using an RTOS: It needs additional RAM and Flash memory and each thread needs an additional stack, increasing the memory need even further. Cost is another factor, but this does not count in our special case: When using Synergy Platform, all is included. And the simple-to-use API of ThreadX® as part of the SSP reduces the steepness of the learning curve a lot!

2.7.2　The Main Features of ThreadX®

Having seen the benefits of using a RTOS, it is time now to look into the main features of ThreadX®. It is small in size, with a very small RAM-memory footprint and is optimized for very fast performance and low overhead, leaving the resources of the microcontroller mostly to the application.

The enhanced real-time scheduling algorithms and efficient multitasking routines provide round-robin scheduling and time slicing, as well as preemptive and Preemption-Threshold™ scheduling. To prioritize threads, there are up to 1024 priority levels available to the software-engineer, with the default number of priorities being 32.

Resources in an embedded system are mainly limited by time and memory resources. ThreadX® provides several powerful options to manage them. Memory needs to be allocated fast and in a deterministic manner, so the SSP provides the ability to create and manage any number of pools of fixed-sized memory blocks. As the pools are fixed size, memory fragmentation is not a problem, but the size of a memory block must be the same or larger than the largest memory request from the application, or the allocation will fail. But making it large enough for that scenario might waste memory if requests come in with different block sizes. A workaround for that is the creation of several memory block pools that contain different sized memory blocks. This disadvantage is outweighed by the fast allocation and de-allocation of these blocks, as these allocations are done at the head of the available list. This provides the fastest possible linked list processing and might keep the actual memory block in cache.

ThreadX® does not only allow block pools with fixed block sizes, but also the creation of multiple byte pools, which behave similar to a standard C heap. Memory is only allocated with the desired size in bytes, based in a first-fit manner. The disadvantage of that is that, like heaps in C, byte pools get easily fragmented, creating a somewhat un-deterministic behavior.

Other resources managed by the RTOS are the application timers, allowing an unlimited number of software timers to be created. These application timers are available in three operation modes: one-shot, periodic and relative. A one-shot timer will call a user-function only once after the timer expires, while a periodic timer calls a user-function repeatedly after a fixed interval. The relative timer is a single continuously incrementing 32-bit tick counter. All the timer expirations are specified in terms of ticks, for example, 1 tick equates to 10 ms, but this is of course configurable. The SSP manages activation and expiration without linear searching, reducing the amount of overhead in timer-centric applications like communications and process control.

Synchronization of threads and communication between tasks is another big topic in embedded systems. ThreadX® provides several mechanisms for that and makes this task very easy and straightforward. Semaphores and mutual exclusions (mutex) help to prevent priority inversion, and allow an unlimited number of objects to be created. Sophisticated callback functions and Event-chaining™ are available as well, as is optional priority inheritance. In addition, ThreadX® can suspend in either FIFO or priority order, avoiding the problem of threads starving for processing time.

Event flags are another thread synchronization feature. An event flag group consist of 32-bits, where each bit represents a different logical event and threads can wait on a subset of these bits. Event flags also support Event-chaining™. As with semaphores, an unlimited number of objects can be created and information on the run-time performance of the flags is available.
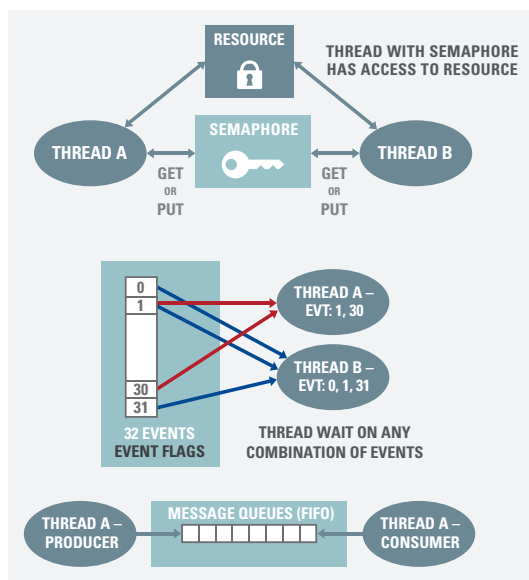

Figure 2-7: ThreadX® synchronization and communication features

The last synchronization feature to be mentioned here is message queues. Using the producer-consumer model for inter-task communication, ThreadX® supports messages sizes from 1 to 16 four-bytes words, with mailboxes being a special case of a message queue with the size 1. Application threads can register a callback function for further notifications. As with the event flags, run-time performance information is available.

For debugging purposes, a very important feature is implemented: A built-in event trace capability supported by the workstation based visualizer TraceX®, which allows to view the sequence of the different events from inside e² studio.

And finally, ThreadX® is compliant with Misra-C:2004, Misra-C2012, and is pre-certified for the following standards:

- IEC 61508
- IEC 62304
- UL 60730-1 H
- CSA E60730-1 H
- ISO 26262 ASIL D

- IEC 60730-1 H
- UL 60335-1 R
- IEC 60335-1 R
- UL 1998
- EN 50 128 SW-SIL 4

Points to take away from this chapter:

- The Synergy Software Package (SSP) is built in layers and all of them can be accessed by simple API-calls.
- The Application Frameworks remove the need to write basic code.
- ThreadX® and the X-Ware™ middleware are well proven components making the creation of full-featured applications a snap.

3 AN INTRODUCTION TO THE APIs OF THE SYNERGY SOFTWARE PACKAGE

What you will learn in this chapter:

- What the different layers of the Synergy Software Package are and how they can be accessed.
- Details about the Application Programming Interfaces and how to use them.

Ease of use was in the mind of the designers while working on the Synergy Software Package (SSP). While offering tremendous functionality, the SSP is quite simple to use, as the design of the Application Programming Interface (API) is very straightforward and comprehensive, encapsulating the complexity of the SSP, but still giving the programmer full control of the functionality. Even programming a complex task like a USB-transfer is reduced to a couple of lines of code and can be achieved without having to read tomes of manuals or to study each and every detail of the register set of a given microcontroller peripheral. This is a huge relief for developers, as they can concentrate on the feature set of their application instead of writing low-level code, which does not add any value to the design, but which takes a good amount of time to write and test. Let us have a look at some of the details of the API!

3.1 API Overview

Applications can access all the functions in the SSP through intuitive and comprehensive API calls, no matter in which of the layers the required functionality resides. This makes it very easy and straightforward to write code which is easy to understand, simple to maintain and painless to port, for example if a different peripheral of the microcontroller has to be used for a certain task, something which is happening more often during a design these days.
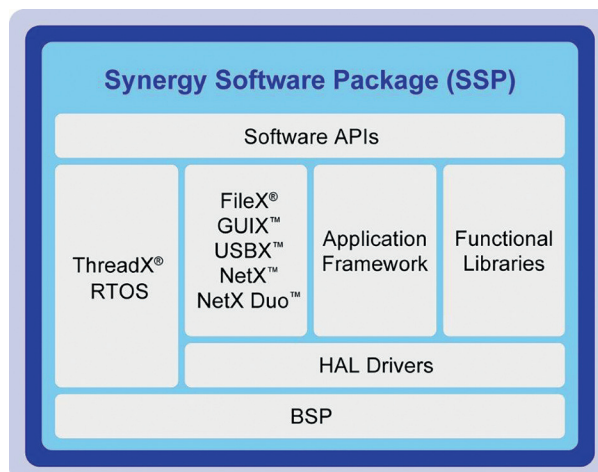

Figure 3-1: The different layers of the Synergy Software Package

Staying with this example, changing from the on-chip SPI-peripheral to the SPI-functionality of the on-chip SCI-port means that there is just one simple change to make if the SSP is used: in the Synergy Configurator the *SPI Driver on r_spi* in the Threads tab needs to be replaced by the *SPI driver on r_sci_spi* driver. As both drivers will use the same instance called *g_spi0*, there is no need to change anything in the application code.

Another possibility is to use the *SPI Framework Device on sf_spi*. In this case, only the *g_spi0 Driver* on *r_spi* driver inside the framework needs to be replaced by the *g_spi0 SPI driver on r_sci_spi* driver inside the Synergy Configurator. All calls to the frameworks API *g_sf_spi_device0* stay the same. There is not even a change in the code necessary. Everybody who, like me, already had in the past the task to change peripherals in existing code, will really appreciate this simplicity offered by the Synergy Platform. This means that making design changes after project start does not create big headaches and are perfectly easy to implement and will also reduce the testing time necessary to a bare minimum.

The architecture of the different layers is shown in Picture 3-1: At the bottom is the Synergy MCU, with the Board Support Package (BSP) sitting on top of it, being responsible for getting the MCU from reset to the main application and providing other services to the software above. The next layer is the Hardware Abstraction Layer (HAL), which insulates the developer from having to deal directly with the register-set of the microcontroller, and which makes the software above the HAL more portable across the entire Synergy Platform.

On top of the HAL are the Application Frameworks, the Functional Libraries and the X-Ware™ middleware, as well as the RTOS. Whilst the Application Frameworks implement commonly used system services like communication or audio playback or capacitive touch sensing, the Functional Libraries contain specialized software, for example for digital signal processing or security related functions. At the very top is the user program, making calls to the layers below through the API. Details on the layers and parts of the SSP are available in chapter 2.

Each of the different layers are accessed through API calls directly, or in a stacked manner. An audio application, for example, can call the Audio Playback Framework of the Application Frameworks, which makes use of the I²C (Inter-Integrated Circuit) driver and two Data Transfer Controller (DTC) drivers to provide read / write access to an external audio converter connected to the microcontroller over the I²C interface, and a General PWM Timer for the time base. Of course, an end application could also access the HAL drivers and the BSP directly, but using the Application Frameworks is much, much easier as no detailed knowledge of the underlying parts is necessary in this case.
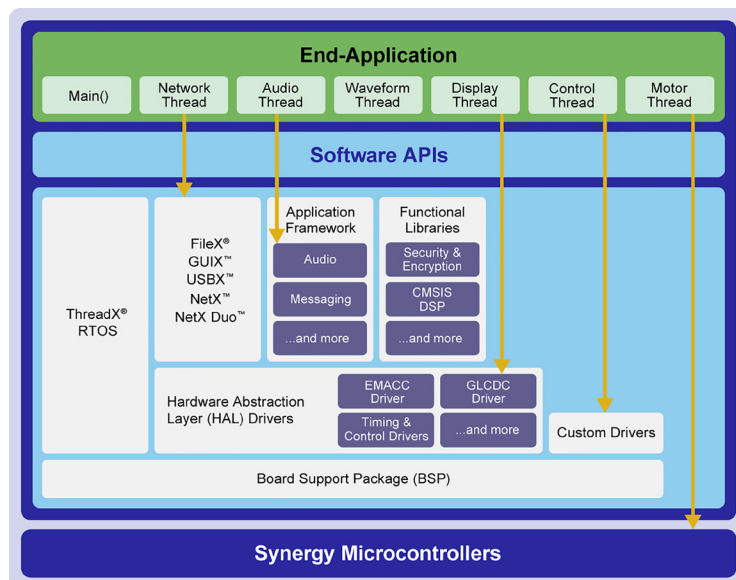

Figure 3-2: User programs can have direct access to the different layers of the SSP, but using the Application Frameworks or the Functional Libraries is much easier

3.2 API Syntax

Before diving more deeply into the details of the API, we should have a short look on the naming conventions of the different APIs and files. Once comprehended, not only programming an application becomes an easy task, but also understanding either code from another programmer or one's own code after several months, an effort which shouldn't be underestimated (something we always do) and where the clear structure provided by the SSP will help a lot.

- **R_BSP_xxx:** The prefix for a common BSP function, e.g. `R_BSP_VersionGet()`.

- **BSP_xxx:** The prefix for BSP defines, e.g. `BSP_CODE_VERSION_MAJOR`.

- **SSP_xxx:** The prefix for common SSP defines, mostly error codes, e.g. `SSP_SUCCESS`.

- **g_<interface>on_<instance>:** The name of the constant global structure of an instance that implements the API of the interface. E.g. `g_spi_on_rspi` for an HAL Layer function or `g_sf_spi_on_sf_spi` for an Application Frameworks function.

- **r_<interface>_api.h:** The name of an interface module header file, e.g. `r_spi_api.h`.

- **R_<INTERFACE>_<Function>:** The name of an HAL Layer API, e.g. `R_RSPI_Open()`.

- **sf_<framework>_<function>:** An Application Frameworks level module.

- **SF<FRAMEWORK>_xxx:** An Applications Frameworks Layer API, e.g. `SF_SPI_Open()` or `SF_AUDIO_PLAYBACK_Open()`.

The naming conventions of the X-Ware™ middleware are similar. They are built in the following way: *<ID>_<noun>_<verb>*. *ID* is the module, noun is the object in question (timer, semaphore, etc.) and the *verb* is the action to be performed (create, close, receive, ...). For example `tx_queue_create()`, which creates a message queue in ThreadX®.

Here is a summary of several of the IDs for the X-Ware™ middleware and the ThreadX® real-time operating system:

- **tx:** ThreadX® related functions, e.g. `tx_thread_create()`.
- **nx:** NetX™ related functions, e.g. `nx_packet_copy()`.
- **nxd:** NetX Duo™ related functions, e.g. `nxd_ipv6_enable()`.
- **gx:** GUIX™ related functions, e.g. `gx_display_create()`.
- **fx:** FileX® related functions, e.g. `fx_directory_create()`.
- **ux:** USBX™ related functions, e.g. `ux_device_stack_initialize()`.

Besides understanding the formal syntax of the API, it is also very helpful to agree on a couple of definitions. All too often, every one of us has a slightly different comprehension of some terms, causing confusion. So, here is the list of terms used throughout the book:

- **Modules:** Modules can be peripheral drivers, pure software, or anything in between, and are the building blocks of the SSP. Modules are normally independent units, but they may depend on other Modules. Applications can be built by combining multiple Modules to provide the user with the features they need.

- **Module Instance:** Single and independent configuration of a Module.

- **Interfaces:** Interfaces are the way Modules provide common features. Through them, Modules that adhere to the same interface can be used interchangeably. Think of an Interface as a contract between two Modules, where the Modules agree to work together using the information agreed upon in the contract.

- **SSP Instances:** While interfaces dictate the features provided, Instances actually implement them. Each instance is tied to a specific Interface, using the enumerations, data structures and API prototypes from the Interface. This allows an application to swap out Instances when needed.

- **Drivers:** A Driver is a specific kind of Module that directly modifies registers in the Synergy MCUs.

3.3   API Constants, Variables and Other Topics

As mentioned, modules are the building blocks of the SSP. They can perform different tasks, but all modules share the basic concept of providing functionality upwards and requiring functionality from below, as shown in shown in Figure 3-3. So, the simplest possible application using the SSP consists of one module with the user code on top.
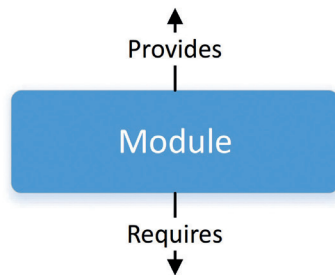
Provides

↑

Module

Requires

↓

Figure 3-3: Modules are built in a way that they provide functionality
to the caller and require functionality from the level below

Modules can be layered on top of one another, building an SSP stack. The stacking process is performed by matching what one module provides with what another module requires. The audio playback framework module, for example, requires a data transfer interface (amongst others), which can be provided by the Data Transfer Controller (DTC) driver module. Its code is not included in the Audio Playback Module by intent, allowing the (underlying) DTC module to be reused by other modules as well, for example by the UART or the SD-card drivers.

The ability to stack modules offers a great benefit, as it ensures the flexibility needed by the application level. If they were directly dependent on others, portability across different user designs would be difficult. The SSP architecture as it was designed provides the flexibility to swap modules in and out (e.g. exchange the UART Driver against the SPI Driver) through the use of the SSP interfaces, as modules adhering to the same interface can be used interchangeably. Remember the contract mentioned above? All the modules agree to work together using the information that was agreed upon in the contract.

On the Synergy MCUs, there is occasionally an overlap of features between different peripherals. For example, I^2C communications can be achieved through the use of the I^2C peripheral or through the use of the SCI peripheral in its Simple I^2C mode, with both peripherals providing a different set of features. The interfaces are built in a way that they provide the common features most users would expect, omitting some of the more advanced features, which however, in most cases, are still available through interface extensions.

At least three data structures are included in each SSP interface: a module depending control structure named `<interface>_ctrl_t*`, which is used as an unique identifier for using the module, a configuration structure named `*<interface>_cfg_t*` as input to the module for initial configuration during the `open()` call and an instance structure, consisting of a pointer to the control structure, a pointer to the configuration structure and a pointer to the API structure. The name of this structure is standardized as `g_<interface>_on_<instance>`, e.g. `g_spi_on_spi` and the structure itself is available to be used through an extern declaration in the instances header file, e.g. `r_spi.h`. While this structure could have been combined in the case of a simple driver, creating them this way expands the functionality of the interfaces.

All interfaces include an API structure containing function pointers for all the supported functions. For example, the structure for the digital-to-analog converter (DAC) contains pointers to functions such as *open, close, write, start, stop* and *versionGet.* The API structure is what allows modules to be swapped in and out for other modules that are instances of the same interface.

Modules alert the user application once an event has occurred through callback-functions. An example of such an event could be a byte received over a UART channel. Callbacks are also required to allow the user application to react to interrupts. They need to be as short as possible, as they are called from inside the interrupt service routine. Otherwise, they might have a negative impact on the real-time performance of the system.
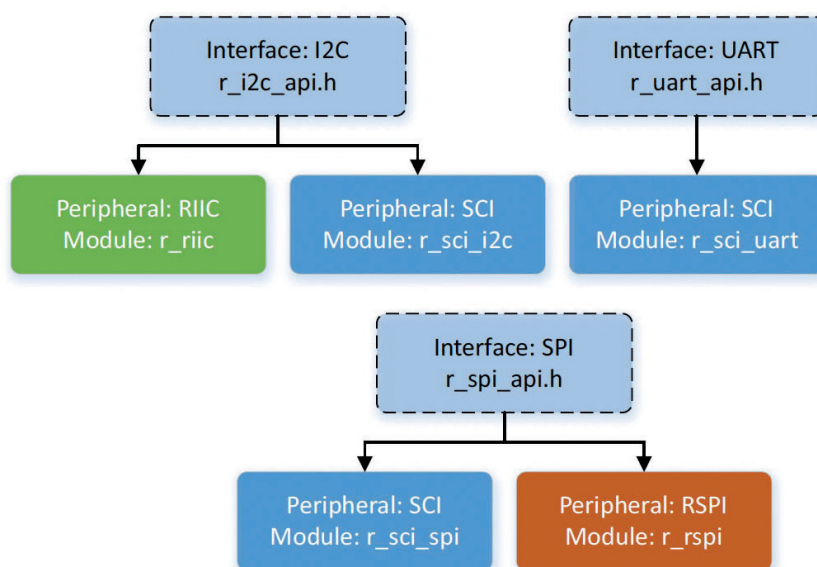
Figure 3-4: On Synergy MCUs some peripherals will have a one-to-one mapping between the interface and instance, while others will have a one-to-many mapping

Whilst interfaces dictate the features that are provided, the instances actually implement them. Each instance is tied to a specific interface and uses the enumerations, data structures and API prototypes from the interface. This allows for applications that use an interface to swap out the instance when needed, saving a lot of time once changes to the code or the used peripherals are needed. On the Synergy MCUs, some peripherals, e.g. the I^2C, will have a one-to-one mapping (it only maps to the I^2C interface), while others, like the SCI, will have a one-to-many mapping (it implements three interfaces: I^2C, UART, SPI). Refer to Figure 3-4 for a graphical representation of the mapping.

3.4   API Usage

After all the theory, it is now time to finally look at how easy it is to work with the Synergy Software Package. For this, we will use a simple UART as example and explain how to use the APIs and where to find information about the different items. If you want to follow this on your PC, you can come back to this section after chapter 5.

The first step in using an SSP module is always to select the right interface for the functionality that is required. In our case, we want to communicate over the UART interface block of the Serial Communications Interface (SCI), which is available on all

series of the Synergy MCU Family, so inside e² studio we select the *UART driver on r_sci_uart* in the Synergy Configurator. This is done by first highlighting the *HAL/Common thread* on the left hand side of the Threads page and then by selecting *New Stack → Driver → Connectivity → UART Driver on r_sci_uart* on the right hand side under *HAL/Common Stacks*. Once added, additional required entries will be added automatically down to a level where the user needs to make a choice or decision on the functionality. In the case of the UART, an instance of each of the *g_transfer0* and *g_transfer1 Transfer Drivers on r_dtc Event SCIO TXI (RXI)* will be added. Once added, UART Driver *(g_uart0 UART Driver on r_sci_uart)* will be shown in red, indicating missing settings.

Hovering over such an item will show the functionality required. In the case of the example above, the receive, transmit and error interrupts need to be enabled in the *g_uart0 UART Driver on r_sci_uart* module section in the Properties-window of the driver. If wanted, the module instance, which is called `g_uart0` by default (the number depends on the selected physical interface), can be given a user name if desired (e.g. `my_uart`). This adds another layer of abstraction, which, if the name is properly chosen by the software designer, can make the code more portable and better maintainable, not to speak of being more readable as well.

The *Properties*-window is also the place to configure the parameters, like baud rate, parity, etc. of the serial port. With all the configuration done already in the graphical user interface, there is no need to initialize the port with the correct values manually. This is all done by the Synergy Configurator automatically. Once the interrupts are enabled, nothing more needs to be done on the *Threads* tab and you will notice that the *g_uart0 UART Driver on r_sci_uart* is no longer highlighted in red, as we have made all the settings required.
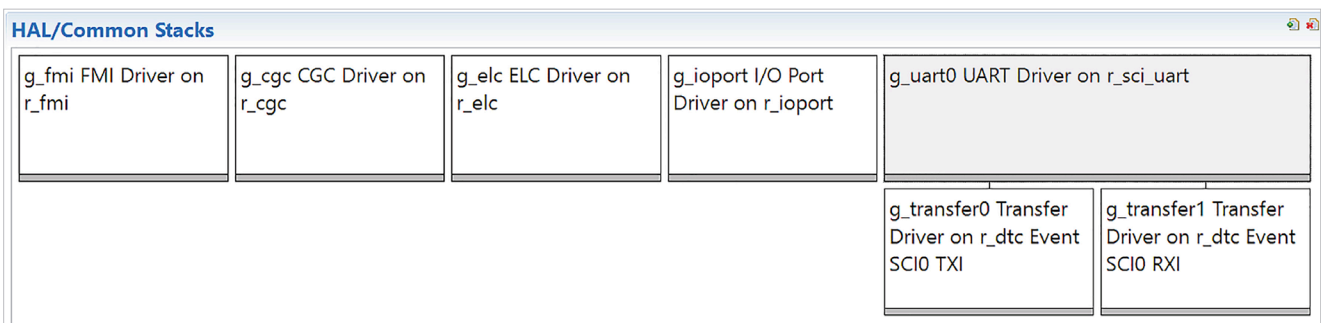

HAL/Common Stacks

| g_fmi FMI Driver on r_fmi | g_cgc CGC Driver on r_cgc | g_elc ELC Driver on r_elc | g_ioport I/O Port Driver on r_ioport | g_uart0 UART Driver on r_sci_uart |

| g_transfer0 Transfer Driver on r_dtc Event SCIO TXI | g_transfer1 Transfer Driver on r_dtc Event SCIO RXI |

Figure 3-5: The UART stack displayed in e² studio

There is no need to care about the other items in the *HAL/Common Stacks* window right now. These are the basic drivers needed by the Board Support Package for the I/O-ports, the Event Link Controller (ELC), the Clock Generation Controller (CGC) and the Factory MCU Information (FMI). The FMI-driver includes a generic API for reading the records from the Factory MCU Information Flash Table, providing information about the features and peripherals implemented on the microcontroller used. All these drivers are added automatically by the configurator.

All that is left is to properly configure the I/O-pins for the SCI UART on the *Pins* tab of the Synergy Configurator to actually transmit and receive data through this peripheral. Expanding the *Peripheral* list on the left side of the view will show the available interfaces. Further expanding the *Connectivity: SCI* tree allows the selection the *SCIO* entry and the *Pin Configuration* for this port will display. Selecting *Asynchronous UART* under *Operation Mode* will configure the pins correctly.

Saving the changes and clicking on the Generate Project Content icon will create the necessary source files and will populate the control and configuration structures, named `uart_ctrl_t*` and `*uart_cfg_t*` respectively, and place them in the application-specific header files to be included in the user code, from where they can be accessed using the pointers provided by the *g_uart0* interface.

With the setup completed and SSP-files generated, interaction with the interfaces gets easy. In case of the UART example, the UART peripheral would be opened and configured by calling the following function:

```
err = g_uart0.p_api->open(g_uart0.p_ctrl, g_uart0.p_cfg);
```

with `g_uart0` being the name given the instance when configuring it in the properties window and `p_api, p_ctrl` and `p_cfg` being the pointers to the structures generated by the configurator. Similarly, writing to the UART would require the function call below:

```
err = g_uart0.p_api->write(g_uart0.p_ctrl, p_src, numBytes);
```

with `p_src` being the pointer to a user defined array holding the data of the type `uint8_t` to be sent over the UART and `numBytes` being the number of bytes to be written as a 32-bit unsigned integer. The exact syntax can be extracted from the HAL interfaces reference chapter in the SSP User's Guide or by using the smart documentation feature of e^2 studio. This short example is a very good demonstration of the capabilities of the Synergy Software Package. Only two lines of code have to be written to configure the port and to send an array of data through it. Everything else has already been created and performed by the Configurators and the drivers of the SSP. No need to read through the manual of the microcontroller and learn about the different registers involved and I believe that this saves a lot of time during the development of an application.

Points to take away from this chapter:

- The SSP is built in layers.
- Each layer can be accessed directly through Application Programming Interfaces (API), but users are encouraged to use the APIs of the Application Frameworks and theFunctional Libraries instead of calling the lower levels directly.
- Using the Application Frameworks and the Functional Libraries makes the code more portable and maintainable.
- The syntax of the API is straightforward and intuitive.

4 GETTING THE RENESAS SYNERGY™ PLATFORM TOOLCHAIN UP AND RUNNING

> **What you will learn in this chapter:**
>
> - Where to get and how to install the toolchain(s) for the Synergy Platform.
> - Steps to be performed during the first startup of the Integrated Solutions Development Environment (ISDE).

In this chapter we will download and install the two Tools necessary to work on software projects for the Synergy Platform: The e^2 studio ISDE and the Synergy Software Package (SSP).

The following paragraphs will outline all the steps necessary and you will find that the installation of both tools is free of any hassle as the installers will take care of everything needed, once you made your selections on what and where to install. Once the installation is done we will perform a short sanity check to ensure that the installation is working.

Just in case you prefer to work with the IAR Embedded Workbench® for Renesas Synergy™ (IAR EW for Synergy) you will find an adapted version of this chapter on the Web.

4.1 Introduction to the Synergy Gallery

First step of the installation process is to go to the Synergy Gallery on the web at https://synergygallery.renesas.com, which is the convenient single point of access to everything needed for the development of a Synergy application: Development environments, compilers, tools, documentation and support. The complete ecosystem is there at your fingertips. And not only from Renesas, but also from Express Logic, IAR Systems® or from the creators of Verified or Qualified Software Add-Ons (QSAs).

To get access it is necessary to register as a user. Registration is straightforward with only a few questions to be answered before access to the site is granted. During the registration process you will also have the opportunity to register your company or organization. This will give you additional benefits as the Company Super User will become available to you, allowing you for example to create any number of software licenses for development and / or production. Note that the first user registering a company becomes the Super User. If your company is already registered, ask your Company Super User to add you to the user database and you are already settled. If you are not sure which registration to use, no worries, as the company registration can be done later on as well. No matter if you register as individual or as company, you will always have full access to the site.

Once registered, a simple log-in will be enough to browse the site and to download any software package immediately. And, everything is accessible from the start-page, so no switching back and forth between different pages is necessary. And just in case, you will need some help or you have a question the web site doesn't answer, there are plenty of options to contact Renesas for help.

The easiest way to get almost instant support is using the Live Chat feature on the support page. Synergy specialists from Renesas will provide level one support online, 24 hours per day, 5 days a week. And this is especially great, as you will not talk to a call-center somewhere on this planet, but to a real expert on the Synergy Platform, knowing it by heart. Other possibilities are to search the Knowledge Base, or to post more complex questions to the Renesas Rulz forum, discussing it with other Synergy developers. On the support page, you can also access documentation, the Synergy Xplorer or browse the Synergy Wiki.

4.2   Downloading and Installing e² studio and the Synergy Software Package

The e² studio contains all the tools necessary to create, compile and debug projects for the Synergy Platform. It is called ISDE, as additional solution-oriented components and plug-ins have been added, making it more powerful. This is especially true for the configurators, which allow an easy graphical access to the different hardware features like the interrupt control unit (ICU) or the clock module without the need for deep study of the user's guide. These configurators will create all the necessary settings and the initialization code automatically and they include an error-checking feature to detect problematic settings already at design time, saving tons of hours otherwise spent for writing and/or debugging code not adding value to the application itself. At least, I for myself would have been more than happy, if such tool had been available when I first started writing software for embedded processors. It's so much easier with the Synergy Platform!

The next paragraphs will walk you through the different steps of downloading and installing e² studio and the SSP on your Windows® workstation.

4.2.1   Download of e² studio and the SSP

The first step to perform is to download the two installers for the e² studio and the SSP from the Synergy Gallery:

On the Synergy Gallery home page, first select *Development Tools* and *e² studio ISDE* from the menu. The e² studio homepage shows up and clicking the blue *Download* button at the lower right hand side of the picture will get you straight to the *Download Details* page. Once there, refrain from starting the download immediately. First check the minimum PC systems requirements to make sure that the tool will work on your machine, then start the download by clicking on the blue Download-button on the right hand side of the *Latest Version* header. Read and accept the license agreement appearing and the download of the zipped installer will start. While the e² studio installer is downloading, take some time to review the installation instructions found under the Documentation tab. They are helpful later on when installing the ISDE.

Downloading the SSP follows the same basic steps: On any page of the Synergy Gallery select *SSP* at the top which will get you to the homepage of the SSP. Again, select the *Download* button which will get you to the associated *Download Details* page. Check again the minimum PC systems requirements, to be sure they match your system. Once more, the blue download button at the right of the *Latest Version* header will start the download process. As with the e² studio download, take some time to review the installation instructions found under the *Documentation* tab and to download the Release Notes and the User's Manual for the SSP.

4.2.2   Installing the tools

Once both downloads are completed, the installation process can start. There should be two files in the download-folder by now: One is called *setup_e2_studio_5_3_0_023.zip* (the revision number may vary) containing the installer for the e^2 studio and the other one is called *SSP_Distribution_1.2.0.zip* (again the revision number may vary) containing the installer for the SSP.

The installation-sequence is important: The e^2 studio ISDE needs to be installed first to allow correct integration of the SSP later on. To start the ISDE-installation, the downloaded archive needs to be decompressed first. This will create a file named *setup_e2_studio_5_3_0_023.exe*. Double-click on it to start the setup. If you are running Windows 10, and depending on the security settings of your workstation, installation might stop with a warning from the operating system that *"Windows protected your PC"*. In this case, click on *"More info"* and on *"Run anyway"*, and it will proceed. Windows® 7 and 8 may produce a similar warning, just click *Yes* and the installation will proceed.

Follow the dialogs of the installer. If your workstation is behind a proxy, you may want to configure it at the *Welcome-*screen (lower right link) to ensure that the installation will work as expected. On the next screen the installation path needs to be provided. Use the default or change it to suit your needs, but remember your selection. Later on you will need to install the SSP into the same folder. On the next screen, the device families can be chosen. Make sure to select Renesas Synergy™.
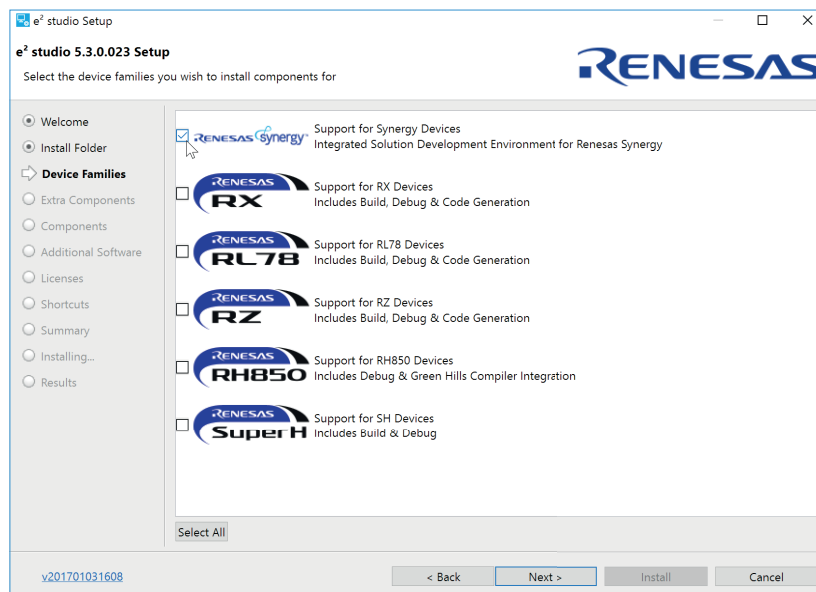
Figure 4-1: Make sure to select the support for Synergy devices

Click on Next and you will be presented with a couple of choices for extra components to be installed. Make your choice. While most options are not particularly useful for Renesas Synergy, you might want to install the support for SVN and GIT. If you do not know yet, don't worry: You can re-run the installer any time later to add or remove features without having to go through the complete installation process again.
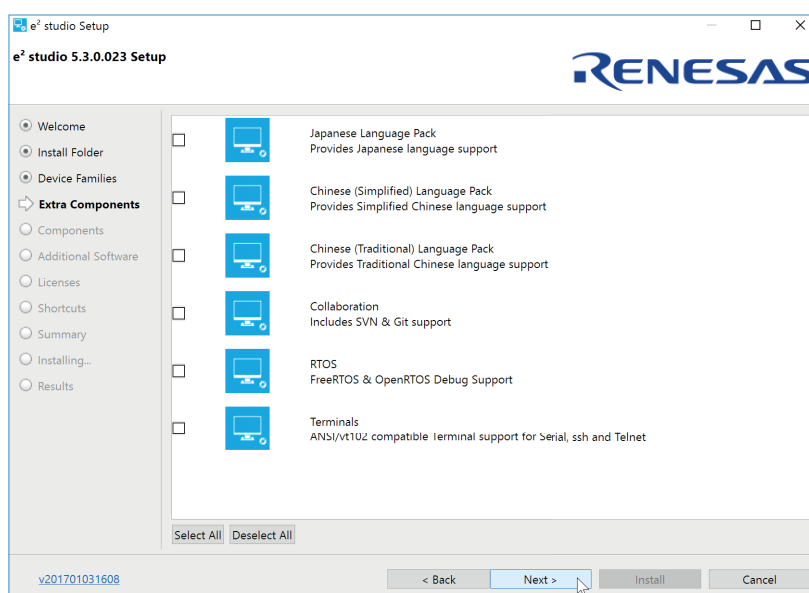
Figure 4-2: Nothing here is essential at the beginning and can be added later

On the following components screen (see Figure 4-3), having everything selected is the way to go. Especially take care not to deselect the *Renesas Synergy RZ Debug Support.* If you do so, you will not be able to create a debug configuration and to connect to a target board.
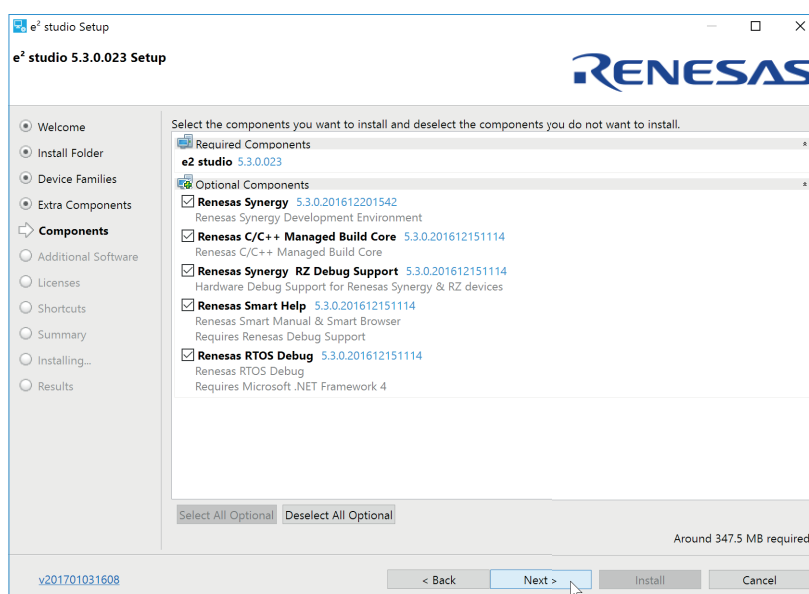

Figure 4-3: You can leave everything selected on this screen

The next screen will allow you to select additional software and the compiler (see Figure 4-4). Check the box beside *GCC ARM® Embedded 4.9 2015q3* (or later revision) to install the code generation tools. Follow the directions on the next screens, accept the terms of the Software Agreements and the setup will start. Be patient, it will take some minutes. Allow the installation of the device drivers from Renesas once prompted.

Later in the process, the installation of the GCC toolchain will start automatically. Follow the directions of the installer and wait for the installation to complete. When asked for the installation path it is highly recommended to leave it at the default, or to write down the exact location. The automatic toolchain integration will only work if it was installed at the default location. If you altered the path, you have to manually integrate it later on.
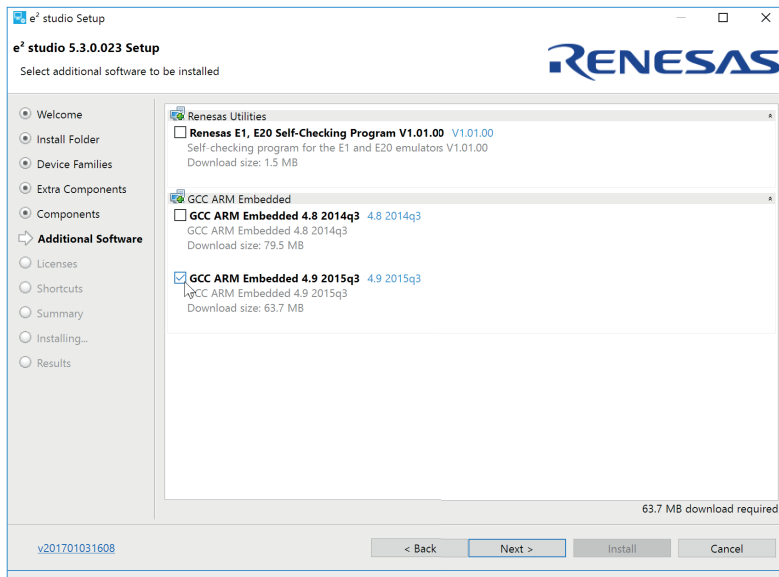
Figure 4-4: It's best to always select the latest compiler version

There will be a last screen, asking for some final steps. Check *Add path to environment variable*, as shown in Figure 4-5. This will add the path of the toolchain to the Windows environment and is necessary to insure an easy registration of the toolchain with e^2 studio. At this point there is no need to run *gccvar.bat*, so you can uncheck this box. Click on *Finish* and later also close the installer of the e^2 studio, once finished.
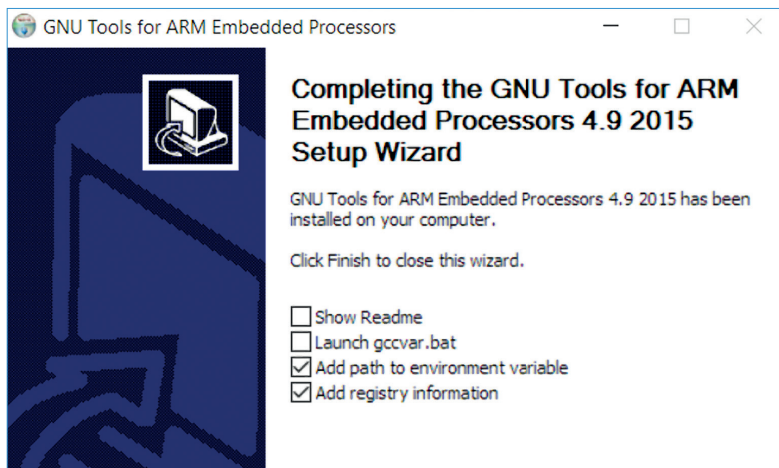
Figure 4-5: It is important to add the path to the environment variable

The next step is to install the SSP. For that, locate the downloaded zip-file *SSP Distribution_1.2.0.zip* and extract and run *SSP_Distribution_1.2.0.exe*. Follow the directions of the installer until you reach the *Choose Components* screen. Make sure that the *SSP Release Pack* and the *SSP Documentation* are selected. On the next screen check the path to confirm that you install the SPP in the root folder of the previously made e^2 studio installation. The ISDE was installed to C:\Renesas\e2_studio by default. If you changed that during the setup of the e^2 studio, you have to enter the correct path here. Click on Install and the installation will start. During the process, you will be asked where to install the documentation. Accept the default, which points to C:\Renesas\Synergy or enter your own path. This will install the following documents: The HTML-version of the SSP User's Manual, a readme-file detailing last-minute updates and a PDF-file with SSP developer examples.

Once this installation has finished it is time to start the ISDE for the first time.

4.2.3   Starting for the first time

Start the e² studio from the *Start Menu* of the PC. It will ask you for a location for the workspace. You can accept the default or choose your own. As this is the first time you start the ISDE, it will extract and refresh additional support files and will restart itself afterwards. During the startup process, you should see a small window called *Toolchain Integration*, as long as you installed the GCC toolchain at the default location. Make sure to select the version of the *GCC ARM® Embedded* toolchain you installed before and click on *Register*. Failing to do so will prohibit compilation of your projects later on and you will have to manually register the tools from inside the ISDE.
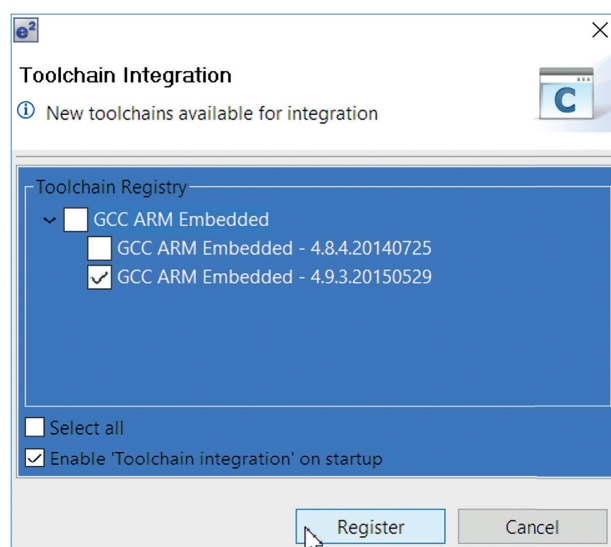
Figure 4-6: Failing to select the installed tool chain here will result in build problems

Once you are at the *Welcome* screen dismiss is by clicking *"Workbench"*. Next step is to create a first simple project to ensure that everything is working. For that, go to *File → New → Synergy C Project* on the menu bar, which will bring up the project configurator. Enter a name for this project, for example *"MySynergyProject"* and select the *GCC ARM®* Embedded toolchain at the right hand side. The next step is to point the e² studio to a valid license file. During the installation of the SSP an evaluation license was placed in the following folder *<e² studio installation path>\internal\projectgen\arm\Licenses*. If you had chosen to accept the default path during installation, this path will be *C:\Renesas\e2_studio\internal\projectgen\arm\Licenses* and the file selector will get you straight to this location. Select the (only) file there, with the name *SSP_License_Example_EvalLicense_20160205.xml* (or similar). Once loaded it will show the license information, the supported components and the related permissions in the window below.

The evaluation license you just installed allows you to use all parts of the Synergy Tools. The only limitation is that viewing or editing the source code for ThreadX®, X-Ware™ components, libraries, stacks, and Application Frameworks API code will be limited. If you want to view or edit this code, you will need a development and production license, which can be requested from the Synergy Gallery.

Click on *Next* to get to the device and tools selection screen. Select a board. As we only want to test the development environment, simply select the *S7G2 SK* for this first test and set the corresponding device *R7FS7G27H3A01CFC* will be listed automatically not already listed. Have a look at the toolchain. It should read *GCC ARM Embedded* with the version *4.9.3.20150529* or later. If you installed the GNU-tools but they are not listed here, you need to integrate them manually. To do this go to the *Help*-menu of e² studio and select the entry *Add Renesas Toolchains.* Please note that the *Scan*

option only works for toolchains having been installed at their default location. If you haven't installed the GNU-tools at their default location, either enter the path to your custom folder in the entry field or browse to the location using the files browser.

Next step is to select a project template. A project template includes the correct device and board dependent Board Support Library (BSP) and, if chosen, ThreadX® support enabled (no matter which template is chosen, ThreadX® can be enabled in the *Threads* tab of the *Synergy Configurator*. In addition, some of the templates include complete example projects as well. For our test, it is sufficient just to use a basic package, so select *Blinky*. Click on *Finish*. The configurator will extract all necessary files and will ask you if you want to open the *Synergy Configuration* perspective. Answer Yes and the perspective for the *Synergy Configuration* will show. In the configurator, you could configure the SSP and most of the on-chip hardware of the processor. For the template we used for this example, everything has already been set, so click on the small green arrow saying *Generate Project Content* at the upper right-hand corner of the window and the configurator will create the configuration files and add them to the project.

Finally click on the small "Hammer"-symbol near the left hand corner of the toolbar on top of the ISDE and the project will compile and link. Once it has finished, it should show a status of 0 errors, 0 warnings after completion.

Now everything is set up and working and you are ready to start your own project. But hold on for a moment and ponder for a few seconds what we just did. I would think that for setting up this enormous software environment for a complete family of microcontrollers, the process is very easy and straightforward. Remember, we installed a software package with over 200 man-years of development effort and a complete ISDE in less than 10 minutes. I consider this as a very effortless process!

And, if something went unexpectedly wrong during the installation, you know help is always available and you can start to chat with a Renesas Synergy™ Platform expert even right out of e² studio: Go to *Help → Synergy Helpdesk* and you will be directed to the Live-Chat on the Internet. Another possibility is to click on the *Live Chat* icon on the *Summary* tab of the *Synergy Configuration* perspective. This will open the chat window directly inside e² studio.

4.2.4   Keeping Your Installation Up-To-Date

Having installed the tools, you might want to keep them in good shape, which means you want to keep them up-to-date. Or you are so excited about all the possibilities the Eclipse-based ISDE offers, that you want to install additional software. Keeping the installation current is achieved by loading the latest installations from the Synergy Gallery. Using the *Check for Updates* feature inside e² studio is not recommended, as the updates there do not add anything useful for Synergy Platform.

If you want to install additional software, go to *Help → Install New Software.* On the following screen, choose the *All Available Sites* to search all sites and you will be presented a list of software available for download. And, of course, another way to get updates is to go to your favorite website, the Synergy Gallery, where you can find updates and additional software as well. For updates to the SSP, the Synergy Gallery is the best place.

Now that we know that the installation is working, the next steps will be to check out a development kit and to write your very own code.

4.3   Installing the IAR Embedded Workbench® for Renesas Synergy™

Installation of the IAR Embedded Workbench® for Renesas Synergy™ follows mainly the same steps. The complete chapter for this is available on-line on the books homepage as a PDF-file for your reference.

Points to take away from this chapter:

- Your single point of access to download the tools is the Synergy Gallery.
- Installation is straightforward, but care should be taken to install in the right order.
- If you do not install the GCC tools to their default directory, you will have to register them manually inside the ISDE.

5 WORKING WITH THE DEVELOPMENT ENVIRONMENTS FOR RENESAS SYNERGY™ PLATFORM

What you will learn in this chapter:

- The difference between Perspectives, Views an Editors
- What the different Configurators are for and how they are used.
- How to import and export projects.

Now that we know the details of the Renesas Synergy™ Software Package (SSP) and how to use its APIs, it is time to look into the different development environments available for the Synergy Platform. At the time of writing, there are two environments available: The Eclipse-based e^2 studio from Renesas, which will be covered in chapter 5.1 and the IAR Embedded Workbench® for Renesas Synergy™, covered in chapter 5.2.

5.1 The Eclipse™ Based e^2 studio

Renesas' own development environment, e^2 studio, is based on Eclipse™, a popular and widespread Integrated Development Environment (IDE) for different programming languages and target platforms. It can easily be customized and extended and is therefore the development environment of choice for thousands of developers worldwide and a de facto standard. And, it will get be updated regularly to use the latest and greatest Eclipse SDK and CDT tools.

e^2 studio leverages all the benefits of Eclipse and includes additional views and configurator perspectives to support all the features of the Synergy Platform. It contains every tool necessary to create, compile and debug projects of any size and complexity and guides the developer through the three phases of a software design: preparation, build and debug.

The following parts of this chapter will talk about the details of the Eclipse workbench and how to use the its different elements of it. While we will cover some ground, not everything will be explained in here. More details can be found in the Workbench User Guide by going to *Help → Help Contents* inside e^2 studio.

5.1.1 Short Introduction to the Philosophy of Eclipse

The main window of e^2 studio, called the workbench, is created of from a few basic user interface elements, like perspectives, views, editors, menu- and tool bars.

Once e^2 studio is started, it opens the perspective(s) last used. A perspective is a set of views, editors and toolbars, together with their arrangement inside the workbench. If you re-arrange windows, toolbars or views, these changes will be saved in the current perspective and are available once you open it again the next time. In e^2 studio, there are several pre-defined perspectives, and you can have multiple several of them open at the same time (like the C/C++ perspective and the resource perspective). Changing from one perspective to another can be done by clicking on the squared icon besides the perspective list at the far right side of the toolbar, which will bring up a pop-up window listing all the available perspectives,

or by selecting *Window → Perspective* from the main menu bar. You can close a perspective using the same menu entry or by right clicking on the perspective in the list.
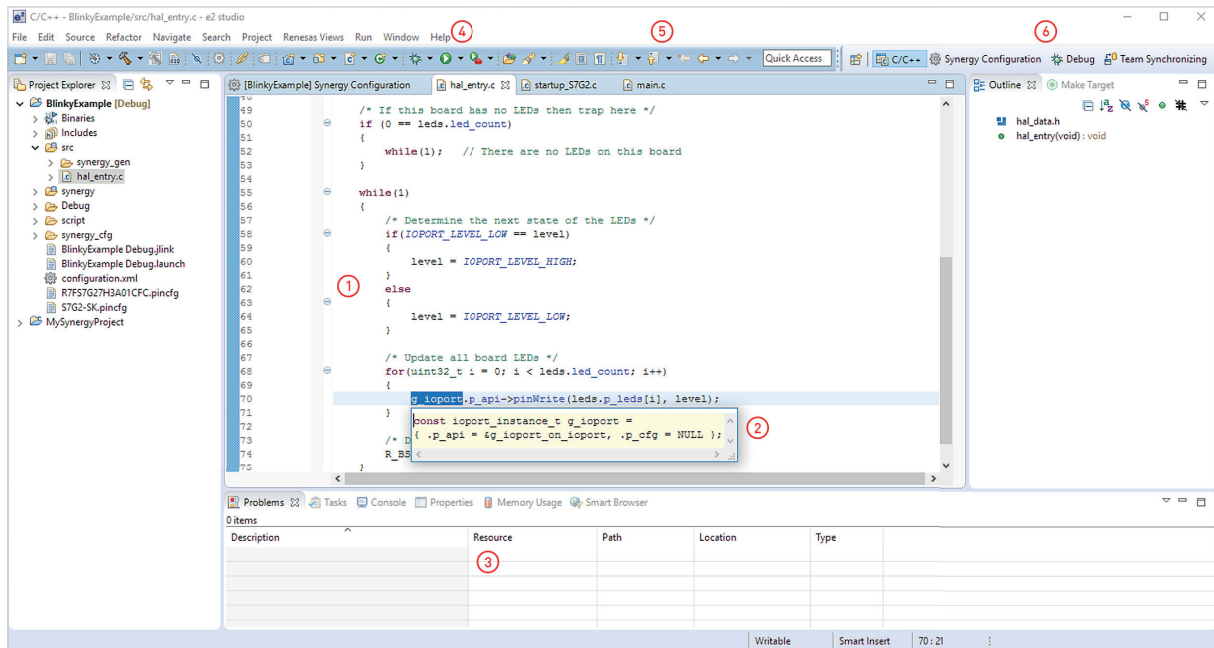

Figure 5-1: The workbench of e² studio consisting of editors (1), Smart Manuals (2), views (3), menu bar (4), tool bars (5) and perspectives (6)

PERSPECTIVES IN e² studio

- **C/C++:** This is the standard perspective of e² studio and is used for developing programs and editing source code. By default, it opens the editor, the Project Explorer view to the left, where you can manage your projects, the Outline view, showing all variables and definitions of the source file active in the editor and at the bottom a tabbed notebook with a couple of stacked views like Problems (e.g. any syntax- or compilation errors), Tasks (shows Doxygen tasks) or Properties.

- **Debug:** This perspective is used for running programs and to diagnose and debug problems that occur at runtime. e² studio opens the following views by default: The Debug view (yes, the Debug perspective includes a Debug view), with the Debug Tool Bar, editor window(s), the Outline view (as with the C/C++ perspective) to the right and some stacked views for analysis and visualization at the bottom and a tabbed notebook on the top for watching variables, breakpoints, registers, and others.

- **Resource:** By default, this perspective includes all the views from the C/C++ perspective, besides the window at the bottom showing the Tasks view.

- **Synergy Configuration:** This is one of the perspectives, where you will spend a lot of time in at the beginning. The largest part of the workbench is occupied by the Synergy Configuration perspective (remember, a perspective can include other perspectives as well), with several views for the different Synergy Configurators, like the Clock Configurator or the Pin Configurator. It also includes the Properties view, where the settings for the different peripherals, threads and drivers can be made, and the Package view, where you see a visual outline of the package chosen for the microcontroller showing the current pin configuration together with a status showing whether a configuration problem has been detected or not. If needed, the Synergy Configurator can be access from the C/C++ perspective and the Debug perspective as well.

- **Team Synchronization:** In this perspective, you can merge changes you had made with those of the other team members. Chapter 5.1.4 will deal with that more in detail.

- **Trace:** This perspective opens Statistics and Histogram views, as well as the ARM® CoreSight™ ITM Live Trace Console, allowing you to trace a given program.

If you change your perspective and you do not like the change, you can always reset it back to the default by right clicking on the perspectives name in the main tool bar and selecting Reset. This also helps to clean up the mess you created after an intensive coding or debugging session, with lots of views and editors open and undocked! No screen is big enough to keep them nicely arranged, so in the end, you will end up with a very confusing workbench.

VIEWS

A view is a window where you can examine something, like a set of registers, properties or a list of files. A view can also have alternative representations, like a real time chart of running threads or gauges and controllers to be connected to variables. Views can have their own menu- or toolbar. Actions triggered by a view's menu- or toolbar will only affect items within that view, but not in other views, even if they reside in the same perspective.

Multiple views can be stacked together in the same window, which is then called tabbed notebook. You can also move views from one notebook to another or undock them totally. It's one of the nice features of e^2 studio that you can arrange everything to match your workflow and that last layout of your perspective will automatically be saved for you.

e^2 studio offers a lot of different views. Renesas even created several additional views like the Conflicts View, the Fault Status view or the RTOS Resources view, making the Integrated Solution Development Environment (ISDE) more versatile. All of them can be found on the main menu bar under *Renesas Views*.

EDITORS

This is the view (the editor is considered a special type of view) every software developer will spend most of his or her time with. Besides the usual and expected features like code completion or keyword highlighting, the editor built into e^2 studio also includes also a special feature, called Smart Manual. It eliminates the need to study 1000's of pages of documentation, as you get context-aware help on the SSP and on the microcontroller itself.

Hovering with the mouse over any of the thousands of highlighted words within e^2 studio will bring up a window displaying relevant information. For variables, the declaration will be displayed, for structures and enumerations all the members and for functions, the description, prototype, and parameter details. If a variable is associated with a register of the microcontroller, detailed information about the bit definitions will be displayed as well. The Smart Manual even pulls in relevant application notes and media-rich instructional material if available and displays them in the Smart Browser view at the bottom of the C/C++ perspective.

This feature is very cool and saves a lot of time, as you will not have to switch back and forth between the SSP and Synergy Microcontroller (MCU) manuals and e^2 studio. You get the relevant information where you need it: Right at the point of your mouse-cursor.

```
/* LED state variable */
ioport_level_t level = IOPORT_LEVEL_HIGH;

/* Get LED information for this board */
R_BSP_LedsGet(&leds);
```
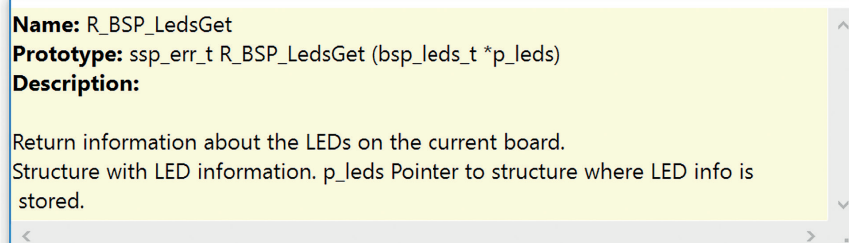
Name: R_BSP_LedsGet
Prototype: ssp_err_t R_BSP_LedsGet (bsp_leds_t *p_leds)
Description:

Return information about the LEDs on the current board.
Structure with LED information. p_leds Pointer to structure where LED info is
stored.

Figure 5-2: The Smart Manual feature of e² studio will display all necessary information directly where it is needed

Code completion is another of those features of the ISDE saving a lot of time during code development. Pressing <ctrl>-space after a variable will bring up a window with the available options, like the members of an API-structure. Clicking on one of the members will insert them into your code. Again, no need to look into the manual, just point-and-click!

5.1.2 Configurators: A Short Introduction

The configurators inside e² studio graphically guide the software developer through specific device options of the Synergy MCU and advises on their use. Examples of that are pin options or warnings of conflicts. The configurators also generate start-up code or place SSP software components within the project.

PROJECT CONFIGURATOR

There are several configurators available to help the designer and the first one most users will experience is the Project Configurator, which leads through the process of creating a new Synergy project from scratch or from a template provided by the configurator. It allows to choose the settings for the project, like the toolchain to be used, which device and board to use or if an example project should be created.

At the end of this process, the project and all necessary files will be automatically created and added to the active project in the workbench of e² studio. There is no need to create anything manually; no looking up of make-file parameters and settings, no research which header files to include, and how to advise the compiler to use a specific series and device out of the large Synergy microcontroller family. All is done for you once you click on *Finish* on the last configurator screen.

With code generation complete, the Project Configurator will switch to the Synergy Configuration perspective, where the settings for the different SSP components can be made.

SYNERGY CONFIGURATOR

The Synergy Configurator first displays a summary of the current project and gives an explanation of the different tabs available and how to use them. The next tab is the *BSP* (Board Support Package) tab, allowing to view and edit aspects of the board setup, like device or board selection. In the associated *Properties* view, additional settings for the BSP can be made, for example the size of the main stack (the stack used outside of a thread context) or the time period of the watchdog timer.

The next tab, named *Clocks*, is intended for setting the initial clock configuration. A graphical representation of the on-chip clock system is shown and changes can be made to the clocking tree. Hovering with the mouse over the items will bring up a short description of them. If incompatible settings are made, the respective member will be highlighted in red and an explanation of the problem will be given. Also, the tab itself changes and displays a small exclamation mark, indicating the presence of a problem.
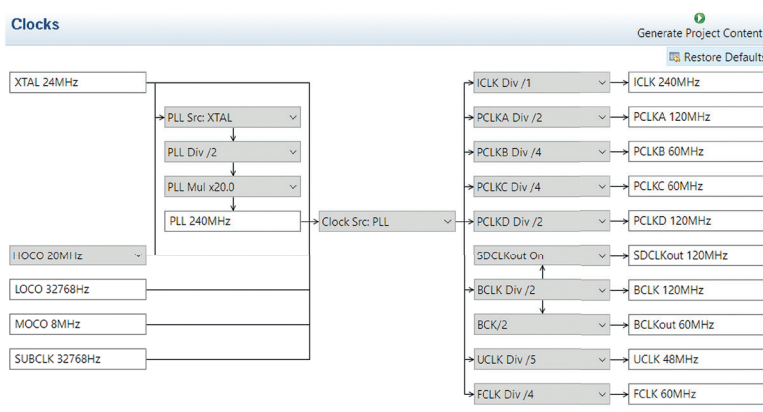
Figure 5-3: The Clocks view of the Synergy Configurator

The *Pins* tab is covering the initial pin setup of the Synergy MCU. Pins can either be listed either based on ports or peripherals. A package view to the right of the configurator shows the package of the device highlighting configured pins and marking errors, if there are conflicts or incomplete settings. These will also be listed in the *Problems*, as well as in the *Pin Conflicts* view.

The tab following, *Threads*, allows you to add and configure the threads within a Synergy project. Different modules and objects can be added to the individual threads and the properties for all of them modified in the *Properties* view. The *Threads* view displays the stacks of the different threads, allowing for a graphical configuration of the modules. New threads can be added easily, with all modules necessary added automatically down to a level, where an intervention of by the developer is required. If this is the case, the module will be marked in red and a description of the necessary settings or problems given, once the mouse is hovering over the module. If resolved, the module turns back to its standard colour.

The messaging framework is configured in the *Messaging* tab, allowing you to create classes of application data messages, and define a list of threads subscribing to each message of interest. Next is the tab called *ICU*, for Interrupt Controller Unit. This tab was previously used to view and edit the available interrupt sources and the associated priority level, but is deprecated (and therefore left empty) in the latest version of e² studio, as the interrupts are now configured through the *Properties* view of the selected module.

The final tab in the Synergy Configuration perspective is named *Components*, allowing you to display and select the different Synergy components. Modifications like adding or removing modules from the current project shouldn't be made in this view, but in the Threads tab, as the latter one will allow the configuration of the modules.

Once all settings are made in the Synergy Configuration, the related source code can be generated or extracted from the SSP and added to the project. This is done by clicking on the *Generate Project Contents* symbol at the top of the perspective. If you forgot to click on it, don't worry! This will also happen if changes to the configuration or generated files are detected.

The Synergy Configurator is a great tool, as it guides you through all the steps needed to prepare a project and to make the initial settings for it. As with the Project Configurator, there is no need to browse through the thousands of pages of documentation and to study them deeply, as the configurator will provide the necessary information on an abstract level and will assure that all settings are correct and plausible. If you remember how much time you spent during your last project not using the Synergy Configurator on just getting, for example, all the pin-routings and interrupt settings right, dealing with a lot of interdependent hardware registers, you will surely appreciate all the effort Renesas put into these new tools from Renesas. It is much faster this way!

5.1.3   Importing and Exporting Synergy Projects

From time to time, you will have a need to import or export projects. Maybe you want to use one of the numerous example projects from the website of Renesas website, or that you want to share your latest and greatest development with one of your peers. No matter if you want to import or export, it can be conveniently done from inside e² studio using either the Import Wizard or the Synergy Export Wizard.

IMPORTING PROJECTS

There are two ways to import projects. Both start at the *File → Import* menu. Once the window of the import wizard comes up, expand the *General* entry you have then the choice to either select *Existing Projects into Workspace* or *Rename & Import Existing C/C++ Project into Workspace.*

In either case, you will be presented with the choice of either selecting a source directory, where the project to import resides, or selecting an archive file. If one or more projects are found, you can select them for import. If you have chosen the *Rename & Import* version, you also have to give the imported project a new name. Click on *Finish* and the project(s) will be imported into your workspace.
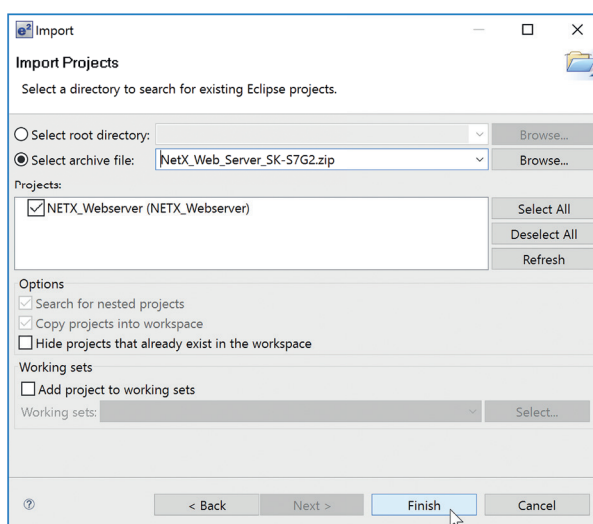
Figure 5-4: The projects found at a given location are listed for import

EXPORTING PROJECTS

Exporting projects is as easy as importing them. Again, there are two ways to achieve the task. The first one is to go to the menu of e² studio and select *File → Export.* The second one is to right-click on the project and select *Export Synergy Project.* If using the first version, there are two additional steps to perform in the export window showing up: The first is to expand the *General* entry and the second to select the *Renesas Synergy Archive File* entry from the list appearing.
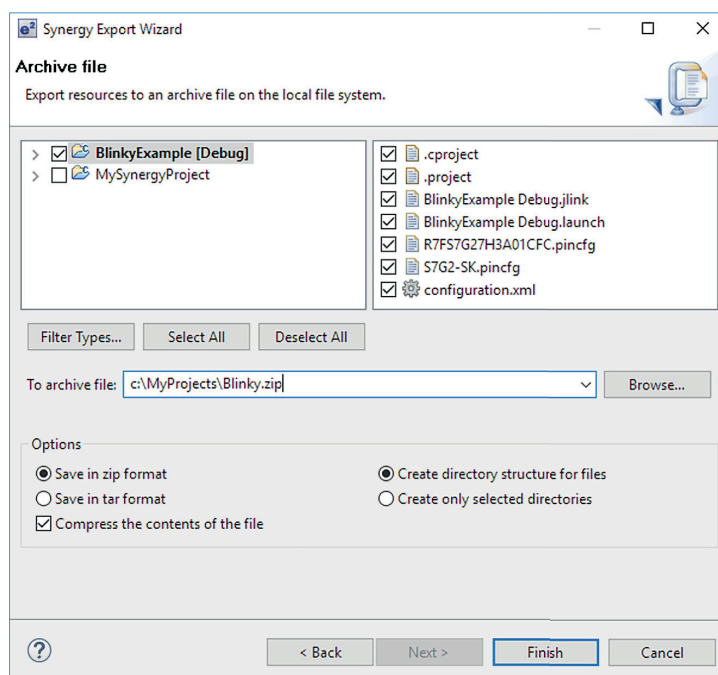
Figure 5-5: The Export Wizard in e² studio

Once the Synergy Export Wizard appears, the projects and files to be exported can be selected, along with the compression format and the desired directory structure. Clicking on *Finish* will finalize the export process.

Do not attempt to export your project by just copying the directory of your workspace, as this will violate the license agreement of the Synergy Software Package, which does not permit You are not allowed to re-distribute distribution of any content from the SSP. And, it is also not necessary to include the SSP files in your export: as the *configuration.xml* file is part of the export, any SSP content can be recreated by opening the file (which will cause the Synergy Configurator to start) and clicking on *Create Project Content* again. This will restore the files omitted during export. A direct copy is also unlikely to work due to multiple hard-references to the project name and location. The Synergy Export Wizard will remove them and the Import Wizard will restore them on import.

5.1.4 Team Collaboration and File Handling

Team collaboration inside e² studio is possible and is easy, as it provides an interface for version control, so you can check-in and check-out files from inside the ISDE. But which parts of the project need to be checked in? The rule of thumb is: Anything created by the Synergy Configurator does not need to be checked in.
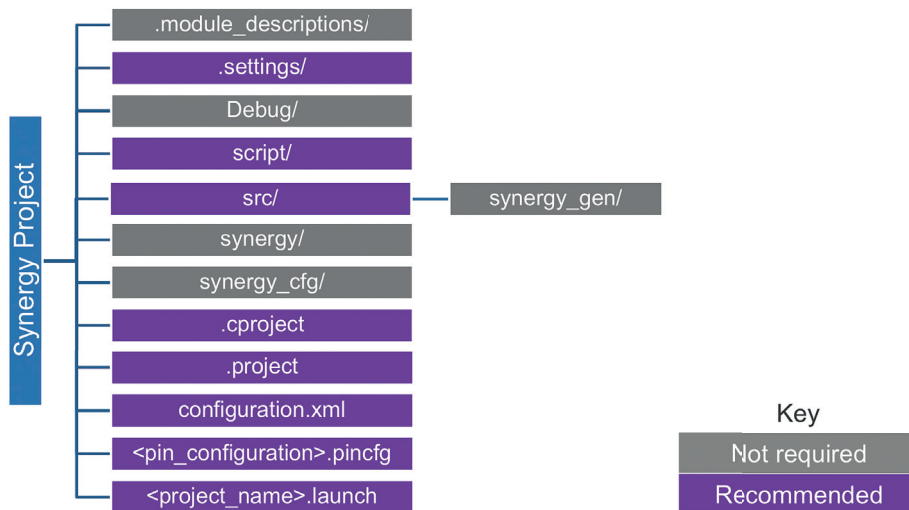
Figure 5-6: There is no need to check-in items in grey

5.2 IAR Embedded Workbench®

As this book mainly covers the toolchain provided by Renesas, this part of the chapter explaining the usage of the IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy) is provided online.

Points to take away from this chapter:

- e² studio provides different perspectives and views to group functionality.
- The Smart Manual feature, the Project Configurator and the Synergy Configurator speed up the development and reduce errors.
- Files created by the Synergy Configurator do neither need to be exported nor checked into a versioning system.

6 RENESAS SYNERGY™ KITS

What you will learn in this chapter:
- What the different hardware kits of the Synergy Platform, their details and their indended use.

There is a point during every development project when you will need hardware to run your first tests. And as all engineers know, this will almost always be the case well before their own hardware is even close to being ready for use. Or maybe the hardware guys don't want to loan one of their rare prototypes to their beloved software colleagues, who may even destroy the board (not that this happened ever to me!)

Renesas has a solution: Renesas offers two kinds of easy-to-use boards which make starting with the platform easy and hassle-free for the software designer, as he can test his programs right away. The hardware designer will also profit from the boards as he can access most of the pins of the microcontroller conveniently at the different connectors of the kits.

6.1 The Different Types of Hardware Kits

Which kit to choose depends on what the designer wants to achieve. The Starter Kit SK-S7G2 offers an easy entry to the Synergy Platform, as it is based on the Synergy microcontroller series offering the largest memory sizes and the largest choice of on-chip peripherals. So it is an excellent choice if it is not clear yet which microcontroller will finally be chosen for the project, or if you have not yet determined which Development Kit is the right one for you.

The Development Kits on the other hand are more intended for a full project prototyping, giving access to all pins and providing additional hardware for specific applications, like a detachable capacitive-touch display. Common to all boards is their modularity and their expandability, as well as their extremely easy and error-proof configuration.

6.2 The SK-S7G2 Starter Kit

The SK-S7G2 Starter Kit is your low-cost entrance card to the entire Synergy Platform and is the starter kit on which this book is based. It allows you to explore the Synergy platform without delay, including the microcontroller and all its peripherals. The compact design gives you access to more than 80% of the pins of the device through Arduino and Pmod™ connectors, allowing for rapid prototyping of your application.

The on-board QVGA touch display makes interaction with the board a snap, especially if used together with the GUIX™ middleware from the Synergy Software Package (SSP) and GUIX Studio™, a Windows program, which can be downloaded from the Synergy Gallery and used to design Graphical User Interfaces (GUI). Connectivity to the outside world is available through USB, Ethernet, RS-232/485 and Bluetooth Low Energy (BLE) 4.1 on the board.

Easy debugging of the software and programming of the device is possible through the on-board J-Link™ debugger. A comprehensive set of documentation is available from the Renesas web site: The Starter Kit SK-S7G2 User's manual, a Quick Start Guide and the board schematics, as well as the S7G2 data sheet and manual.
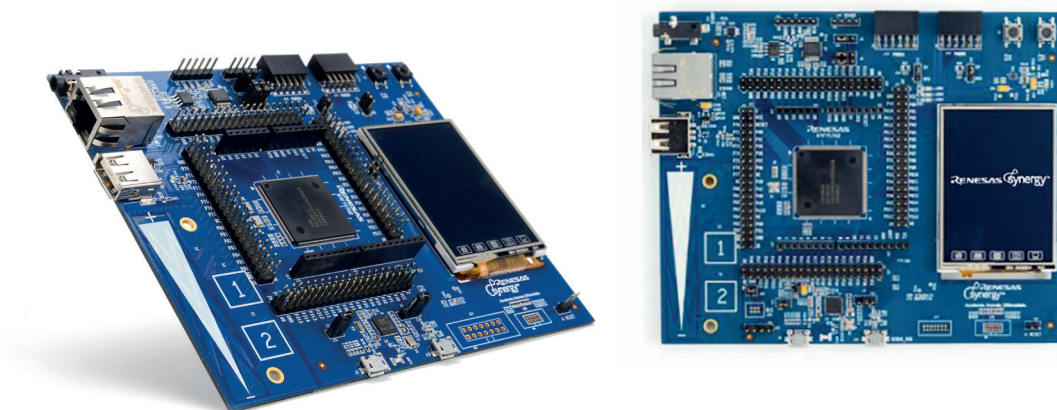

Figure 6-1: The SK-S7G2 Starter Kit

Other features of the Starter Kit include:

- 2.4" TFTLCD QVGA (320 x 240) colour display with touchscreen which uses the internal SRAM of the S7G2 for display frame buffer memory.

- Expansion:
  – Arduino UNO Shield compatible connector
  – 2 x Pmod™ connectors
  – Pin row headers

- Wired connectivity
  – USB (1 x HS Host, 1 x FS Device)
  – Ethernet with RMII and IEEE 1588 Precision Time Protocol (PTP) support
  – RS232/RS485 available on pins with transceiver
  – CAN available on pins with transceiver

- Wireless connectivity
  – Bluetooth Low Energy (BLE) 4.1 on board plus many options through Arduino and Pmod™ expansion connectors

- Memory
  – On-chip: 4 MB flash + 640 KB SRAM
  – External: 8-MB QSPI flash

6.3   Synergy Development Kits

If the SK-S7G2 Starter Kit was your standard entrance card to the Synergy Platform, the Development Kits (DK) are your VIP-pass, as they can be used for complete project prototyping with 100% of the pins available through Pmod™ connectors and through pin row headers.

As most pins on the Synergy microcontrollers support multiple functions, they can be connected to more than one connector or device on the kits. DIP-switches on the PCB allow an easy, safe and error free routing of the different functions. Each DIP-switch controls a high-speed buffer which, depending on the switch' setting, either connects or insulates the MCU-pins from the peripheral or connector. Additionally the DIP-switches can be read by software through the I/O expanders' IIC-port and, if the switch is open, the software can enable the buffers. LEDs will then indicate when the respective device is connected under software control. This is a pretty cool feature as you do not have to move tons of tiny jumpers around, so routing of the signals is really simple.

There are a couple of features which are available on each of the different Development Kits:

- Full access to all the pins and features of the microcontroller.

- Wired connectivity:
    – USB
    – RS232/RS485 through screw-in connectors
    – CAN through screw-in connectors

- Wireless connectivity
    – Bluetooth Low Energy (BLE) 4.0 on board plus many options through Pmod™ connectors

- Several sensors for temperature, acceleration, light, etc.

- Debugging and programming through J-Link® on-board debugger

There are several Development Kits available for different devices and additional peripherals will be available on the PCB (depending on the feature set of the microcontroller used). As with the Starter Kit a full set of documentation is available for each of the boards.

6.3.1   The DK-S7G2 Development Kit

The DK-S7G2 is hosting the S7 Series S7G2 MCU which is today's flagship device in the Synergy Platform. Designed with ease of use in mind, it features all the different things mentioned already and much more. The kit consists actually of several boards: One breakout board carrying different interfaces like Ethernet 10/100 or connectors for the camera board, a main board with the microcontroller, several connectors, LEDs, etc., and two Expansion Boards, one display board carrying a WQVGA TFT LCD and one camera board, carrying an CMOS VGA image sensor.

Unique features of the board in addition to those already mentioned are:

- 4.3" WQVGA (480 x 272) TFTLCD colour display with capacitive touch

- VGA (640 x 480) CMOS image sensor with support for a maximum of 30 FPS

- Expansion through pin-row connectors and four Pmod™ connectors
- Wired connectivity:
    – Ethernet with RMII and IEEE 1588 precision time protocol (PTP) support

- Memory:
    – On-chip: 4 MB flash plus 640 kB SRAM
    – Off-chip: 16 MB SDRAM, plus 16 MB QSPI flash plus 2 GB eMMC

- Full size SD card interface
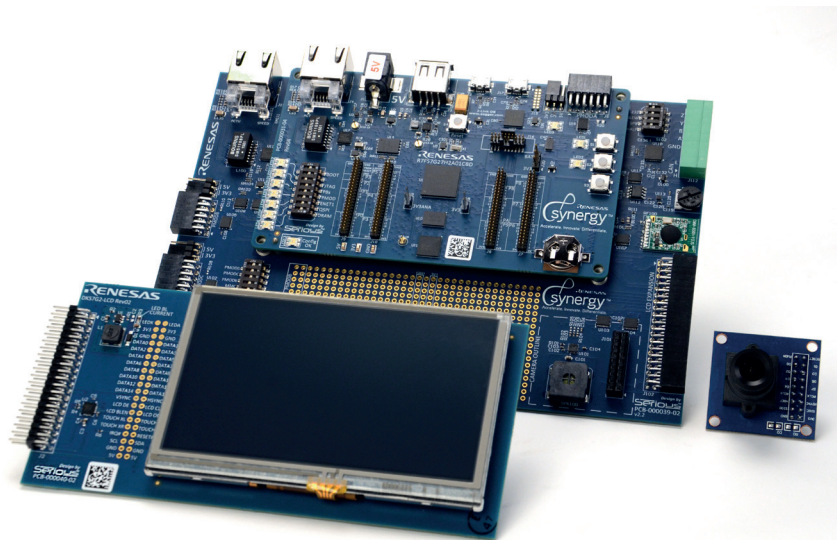
Figure 6-2: The DS-S7G2 Development Kit

6.3.2   The DK-S3A7 Development Kit

The DK-S3A7 features an S3 Series S3A7 MCU, a super-efficient microcontroller inside the Synergy family of microcontrollers. The kit was designed to allow end-product development while measuring the efficiency of the device. The kit consists of three boards: One Main Board with the microcontroller and several connectors and interfaces, one Breakout Board with several peripherals like sensors for light and temperature, as well as Pmod™ connectors and one LCD panel with a 176-segment display.

Unique features of the board in addition to those already mentioned are:

- Detachable 176-segment T6022A-1PRP0 LCD panel
- Expansion through pin-row connectors and three Pmod™ connectors
- Memory:
    - On-chip: 1 MB flash plus 160 kB SRAM
    - Off-chip: 32 MB QSPI flash
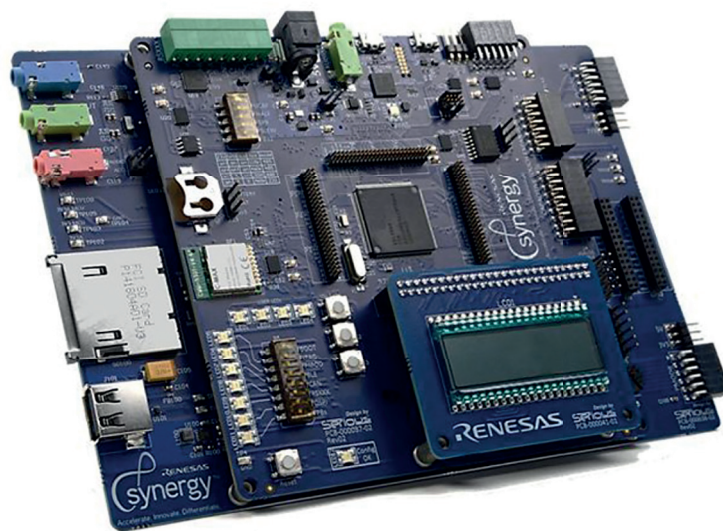
Figure 6-3: The DK-S3A7 Development Kit

6.3.3   The DK-S124 Development Kit

The DK-S124 is hosting the S1 Series S124 microcontroller, a very power efficient MCU with a smart mix of analogue and digital peripherals. Its design allows to make precise measurements of the microcontroller' current consumption in all modes of operation, as well as a detailed estimation of the performance of the devices analogue peripherals. The kit consists of two boards, one main board with the MCU and I/Os and one display board with the following features:

- A small RSK Pmod™ connected display board
- Capacitive touch buttons, one slider
- Expansion through pin-row connectors and one Pmod™ connector
- On-chip 128 kB flash plus 16 kB SRAM
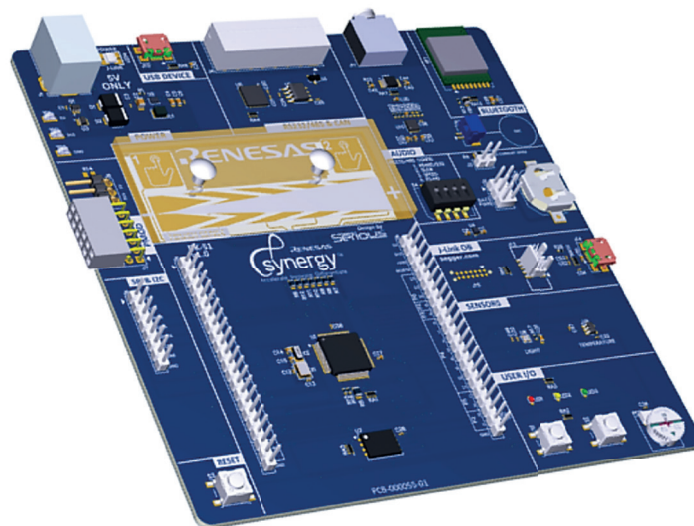
Figure 6-4: The DK-S124 Development Kit

Points to take away from this chapter:

- The SK-S7G2 Starter Kit is your entry to the Synergy Platform, where you can test most of its features.
- The Development Kits can be used for complete project prototyping and each DK includes a different set of external hardware for specific purposes.

7   STARTING THE RENESAS SYNERGY™ STARTER KIT S7G2 FOR THE FIRST TIME

What you will learn in this chapter:

- How to connect the Renesas Synergy™ Starter Kit to your workstation.
- What a debug configuration is and how to create it.
- How to download and start a program in the development environment.

In this chapter, we will test if the Synergy Starter Kit SK-S7G2 is working and communicating with the Windows® workstation and the debugger inside e² studio. For this, we will use the project we generated in chapter 4.2.3. If you didn't do this exercise, don't worry, you can download it from the Website for this book. But in the latter case, you should have already installed and tested the Synergy toolchain as discussed in chapter 4.2.

7.1   Connection and Out-Of-The-Box Demo

If you didn't do that already, it is now time to unpack the SK from its box. Check the contents if everything is there: The SK main board, one USB type A to micro-B cable and the Quick Start Guide. Note that the main board contains a SEGGER J-Link® On-Board (OB), providing full debug and programming capabilities, so there is no need for an emulator when using the SK and the USB cable coming with the kit is fully sufficient.

Next, insert the small plug of the USB-cable into the connector called *DEBUG_USB* (J-19) and the other end into a free USB port of your workstation. The green LED 4 at the top right hand corner of the PCB should light up, indicating that the board has power.

Once the kit is connected, it powers up and performs a self-test, after which a splash-screen will show on its LCD display. Tap on the splash screen and you will enter the pre-programmed thermostat demonstration. For that demo, the Synergy Software Package (SSP) was used to read the internal temperature sensor of the S7G2 Microcontroller (MCU) through its A/D-converter and to display the information.

Play a little bit around with the demo: Tapping on either one of the fields for the temperatures, fan or system will bring up a screen, where you can modify the state of the air conditioning (on/off, cool, heat) or the fan (auto /on) or the temperature setting. Tapping on the Settings icon, which looks like a gear, gets you to a screen where you can make adjustments to the system including units, time and date.

Once you are done with playing around, it is time to move on.
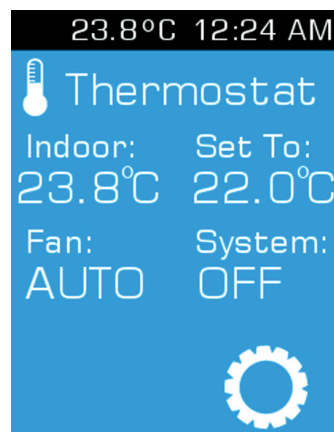
Figure 7-1: Tapping on the different items on the screen will get you to the configuration menus

7.2   Downloading and Testing an Example

With the SK-S7G2 still connected to your Windows® workstation, start the e² studio ISDE from the start menu of the operating system. If prompted for the location of the workspace, use the one you entered during the exercise in chapter 4.2.3 (it should already be listed). If you didn't do the exercise, don't worry, you can download the project from the Website created for this book (www.renesas.com/synergy-book)! Once downloaded, import it into your workspace according to the instructions outlined in chapter 5.1.3. In this case, just use a workspace location of your choice.

Before we can download the program to the kit and run it, you need to create a debug configuration first. Click on the small arrow beside the *Debug* symbol and select *Debug Configurations* from the drop down list.
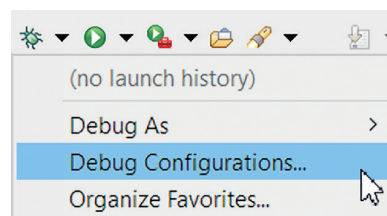
Figure 7-2: To start debugging, select Debug Configurations from the drop-down menu

In the window showing up, highlight *MySynergyProject Debug* under *Renesas GDB Hardware Debugging.* If you used a different name for this project, select the one you used.

Selecting your project will bring up a new screen for the Debug Configuration, showing all the different options for it. No need to change anything in here for our test purpose, just click on *Debug* at the bottom and the debugger will start. If the *Confirm Perspective Switch* dialog displays, click *YES*.

There might be a second dialog named *J-Link® Firmware update* showing up, wanting to install a new firmware version on the on-board debugger. I highly recommend to allow the update by clicking *Yes*.
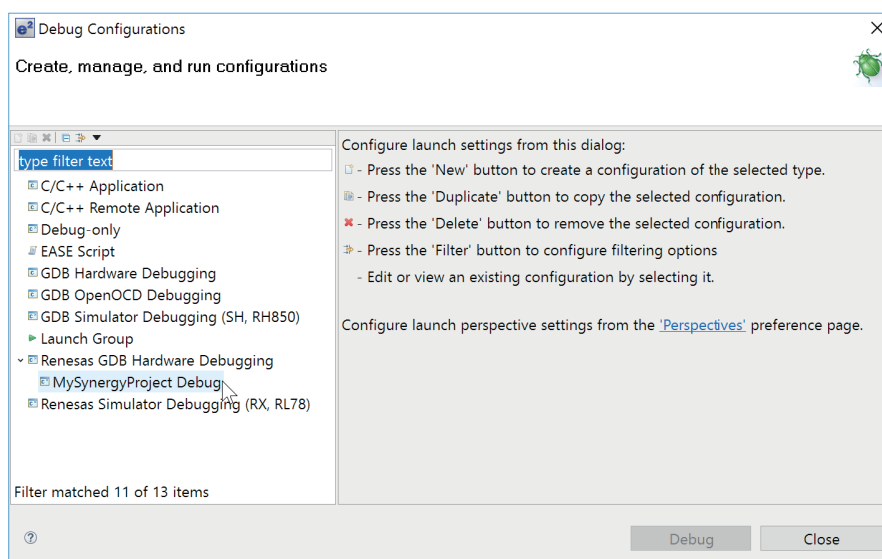
Figure 7-3: Select your project under Renesas GDB Hardware

Once the *Debug* perspective is open, the debugger will set the program counter to the entry point of the program, the reset handler. Click on the *Resume* button ▯▶ and the program will run to the next stop inside the `main()` function, at the line with the call to `hal_entry()`. Click *Resume* again and the program continues to execute, flashing the green, red and orange LEDs (LED 1 through 3) on the Starter Kit in a one second interval.

The final step is to disconnect the debugger from the board by clicking on the *Disconnect* button ⚡ stopping the execution of the program.

Now that you are sure that your installation of e² studio works together with your SK, it is time to write your first Synergy program. This will be the topic of the next chapter.
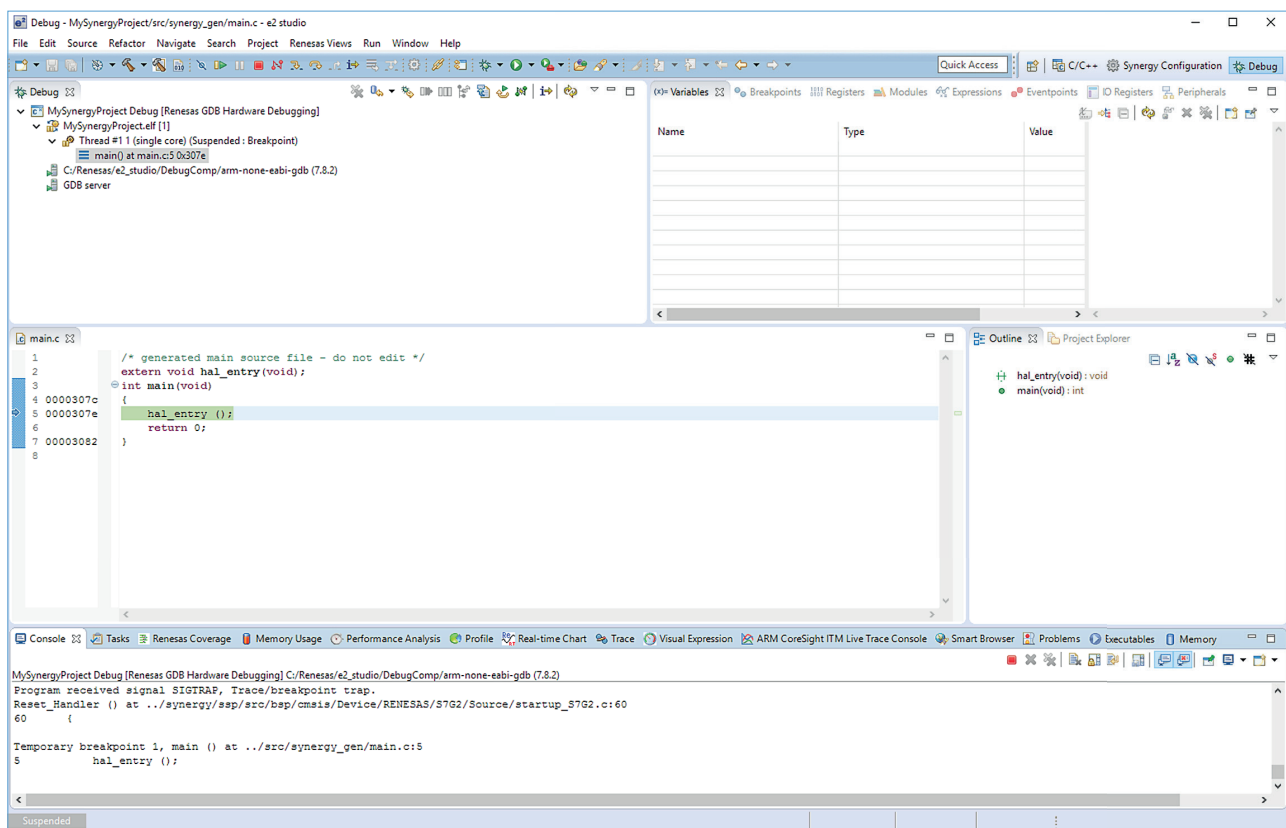
Figure 7-4: The debug perspective of e² studio

Points to take away from this chapter:

- In order to download and debug a program to any hardware, you need to create a debug configuration first.
- Once loaded, the debugger will set the program counter to the entry point. The next stop will be at main().

8   HELLO WORLD! – HELLO BLINKY!

What you will learn in this chapter:

- How to create a project for the Renesas Synergy™ Starter Kit (SK) SK-S7G2 from scratch.
- How to change settings for the Synergy Software Package (SSP) in the Synergy Configurator.
- Write code to toggle the LEDs on the SK.
- How to download and test code

The very first program most newcomers to a programming language wrote (and still write) is the one, which simply puts the string "Hello World" to the standard output device. For me, it was typing "Writeln ('Hello World')" into the editor, as I started with Pascal. Ever since then, I wrote similar lines in several other languages, mostly as a sanity check for the installation of a new development environment.

When I moved on to program embedded systems in the late 1980's, there was no screen where the string could be sent to. So how to instruct the processor to give signs of life? LEDs were barely found in these systems, so toggling one of the very few I/O-pins and observing the waveform with an oscilloscope was the way to go. Over the years, LEDs became a commodity item and we placed plenty of them on our boards, using their blinking as the new "Hello World".

And this is also the objective for this chapter: Toggle an LED on a SK, using everything you learned in the chapters before: You will write the code (nearly) from scratch, create a new project using the configurators, employ the APIs of the SSP, and finally download, debug and run the code. This exercise brings everything together.

As a prerequisite, e^2 studio or the IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy), as well as the SSP should be installed on your Windows® workstation (see chapter 4 for details) and you should have verified that your setup is working (as described in chapter 7). And those of you doing the previous hands-on exercises: Please bear with the author, as he decided to cover again some of the topics already discussed for the sake of those of you moving directly from the foreword to this chapter.

The printed version of the book only covers e^2 studio. If you want to use the IDE from IAR Systems®, you can download the full version of this chapter from the book's Website (www.renesas.com/synergy-book), which includes the same walkthrough for IAR EW for Synergy.

For this exercise, we will again use the SK-S7G2, which is extremely well suited for tasks like this, as it allows you to explore the Synergy microcontroller and all its peripherals instantly. External hardware can be easily attached, as more than 80% of the pins are accessible through connectors. And with the Synergy S7G2 Group MCU being the superset device, every feature of the Synergy MCU Family can be evaluated and results later on applied to the smaller siblings of the Family as well. Figure 8-1 shows the block diagram of the board.
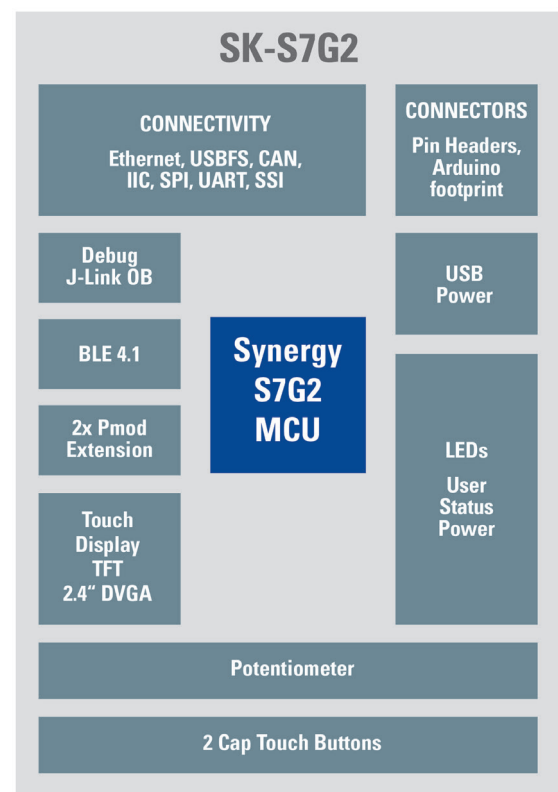
SK-S7G2

CONNECTIVITY
Ethernet, USBFS, CAN, IIC, SPI, UART, SSI

CONNECTORS
Pin Headers, Arduino footprint

Debug J-Link OB

USB Power

BLE 4.1

Synergy S7G2 MCU

2x Pmod Extension

LEDs
User Status Power

Touch Display TFT 2.4" DVGA

Potentiometer

2 Cap Touch Buttons

Figure 8-1: Block diagram of the SK-S7G2

8.1 First Project Using e² studio

8.1.1 Creating a Project with The Project Configurator

If not already done, start e² studio from your Windows® workstation Start Menu. Once the ISDE is up and running, dismiss the Welcome screen, if it shows, as it would block other windows from viewing.

Writing a new program for a microcontroller in e² studio always requires to create new project first, so this is the first step you need to take.

For this, go either to File → New → Synergy C Project, or right-click in the Project Explorer and select New → Synergy Project. Both ways will open the Project Configurator.
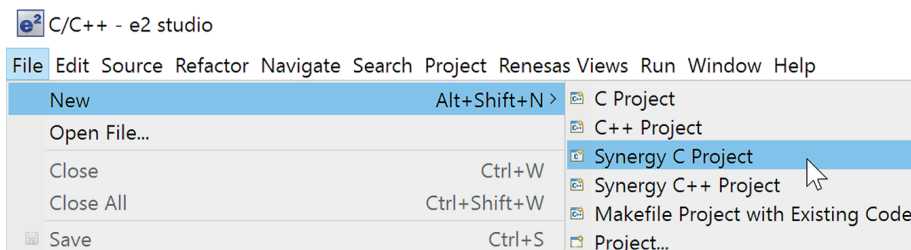
Figure 8-2: First step is to call the Project Configurator

Give the project a name, for example *MyBlinkyProject* and check if the *GCC ARM® Embedded* is highlighted under *Toolchains*. Next, verify that a license for the SSP installed. If the field with the *License Details* is empty, click on Change license file and the dialog window appearing will guide you to the directory where the evaluation license was placed during installation (if you installed e² studio and the Synergy Software Package into the default location, the path will be *C:\Renesas\e2_studio\internal\projectgen\arm\Licenses*). Select the file (there should only be this one) named *SSP_License_Example_EvalLicense_20160205.xml* (or similar). If you already requested and received a development and production license either from Renesas or from the Super User of your company, point the file selector to the correct path for that one. Once loaded, you can examine the permissions for the different components of the SSP inside the *License Details frame*. With this done, click on the Next button to move on to the *Board Selection* screen.

Under Device Selection look out for the field called *SSP version:* It should show the same version of the Synergy Software Package you downloaded before. Under *Board*, select *S7G2 SK*, as this is the hardware we want to use for our small »Hello World« program. Verify that *R7FS7G27H3A01CFC* is shown beside *Device*, it should have been automatically selected. If not, navigate through the drop down list until you found it. In the *Select Tools* frame, verify that *Toolchain, Toolchain version* and *Debugger* read *GCC ARM® Embedded*, *4.9.3.20150529* (or later) and *J-Link ARM®*. These fields should be pre-populated for you. If not, modify them to match the values given above.
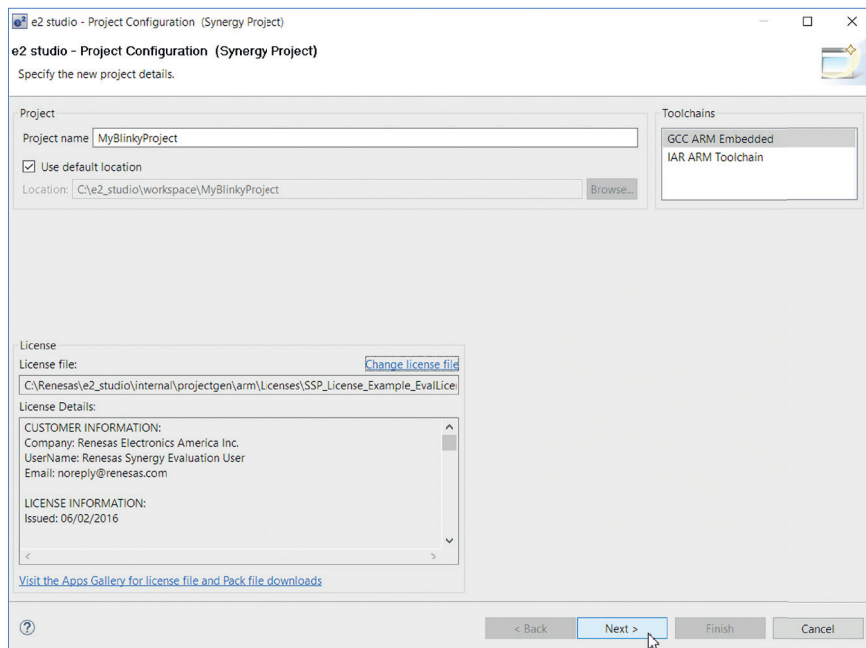
Figure 8-3: The first screen of the Project Configurator mainly asks for the project's name

With this done, click on *Next* to open the *Project Template Selection* screen. A project template may include several items, at least it includes the correct Board Support Pack-age for the selected board / device combination. Some templates even include a complete example project. In our case, select the *BSP* entry, which will load the Board Support Package for the Starter Kit. Click on *Finish*.
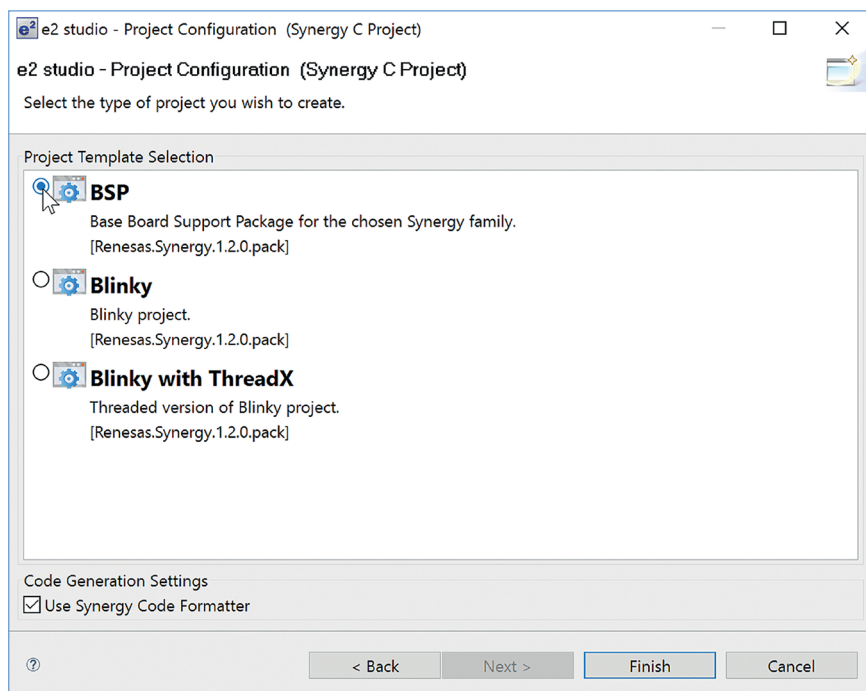

Figure 8-4: Selecting the BSP entry will load the correct Board Support Package

The *Project Configurator* will close and will create all the necessary files for the project in a last step. Once this post-processing is complete, you will be asked if you want to open the *Synergy Configuration* perspective. Select *Yes*.
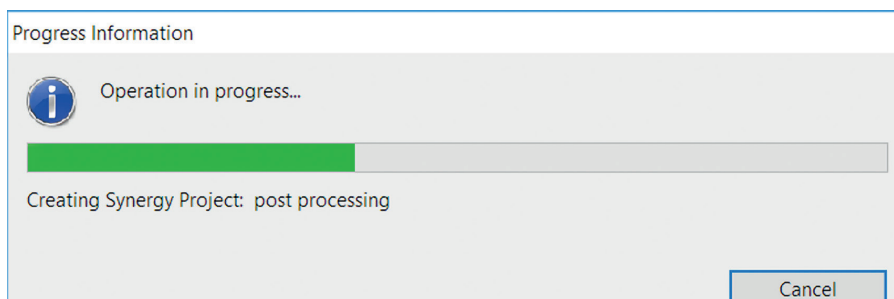
Figure 8-5: The post-processing step of the Project Configurator will create the settings and files for our project

8.1.2   Setting Up the Runtime Environment with The Synergy Configurator

Once the Synergy Configurator has started, it presents you a summary of the project and a short overview over the Synergy microcontroller selected. It also provides a convenient shortcut to the online-chat, which allows you to contact one of Renesas' Synergy specialists directly. The following tab, called *BSP*, allows you to view and edit aspects of the board setup, while in the *Clocks* tab, the initial clock configuration for the project can be set. Any potential problem will be highlighted in red and hovering with the mouse over a highlight will display an explanation. The third tab in the configurator, called *Pins* is covering the initial pin setup of the Synergy device in the project. Pins can be listed either based on ports or peripherals. A package view, again with error marking, is available just at the right hand side of the configurator, reducing possible mistakes to a minimum. The *Threads* tab, allows to add and configure different components. Since we do not use an RTOS in this project, there is only a *HAL/Common* entry, showing the necessary basic modules. Next is a tab called ICU, for Interrupt Controller Unit. This tab was previously used to view and edit the available interrupt sources and the associated priority level, but is deprecated (and therefore left empty) in the latest version of e^2 studio, as the interrupts are now configured through the Properties view of the selected module. The final tab *Components* displays the elements available in the SSP and which of them are currently included in the project. It also allows to switch between different versions of the same module, in case you have multiple versions of the SSP installed. Modifications like adding or removing components shouldn't be made here, but in the *Threads* tab, as they can be configured there as well.

For our project there is no need to change anything in this configurator, as all necessary settings have been done for us already by the *Project* configurator. As the final step in the Synergy Configuration, additional source code based on the current configuration needs to be created. Click on the *Generate Project Content* button at the top right hand side of the Synergy configurator. With that, the required files will be extracted from the SSP, adjusted to the settings made in the configurator and added to the project.

8.1.3   Writing the First Lines of Code

With all the automatically generated files being now in place, it is time to have a look on what was created. The *Project Explorer* at the left hand side of the ISDE lists everything currently included. The *src* folder contains a subfolder called *synergy_gen*, holding configuration sets such as channel number etc. The *src* folder also includes a file named *hal_entry.c*. This is the one you will edit later on. Please note that while there is a file called *main.c* in the *synergy_gen* folder, your user code must go to *hal_entry.c*. Otherwise you will loose your changes, if you make modifications in the Synergy Configurator and re-created the project contents, as this file will be overwritten each time you click on *Generate Project Content*.
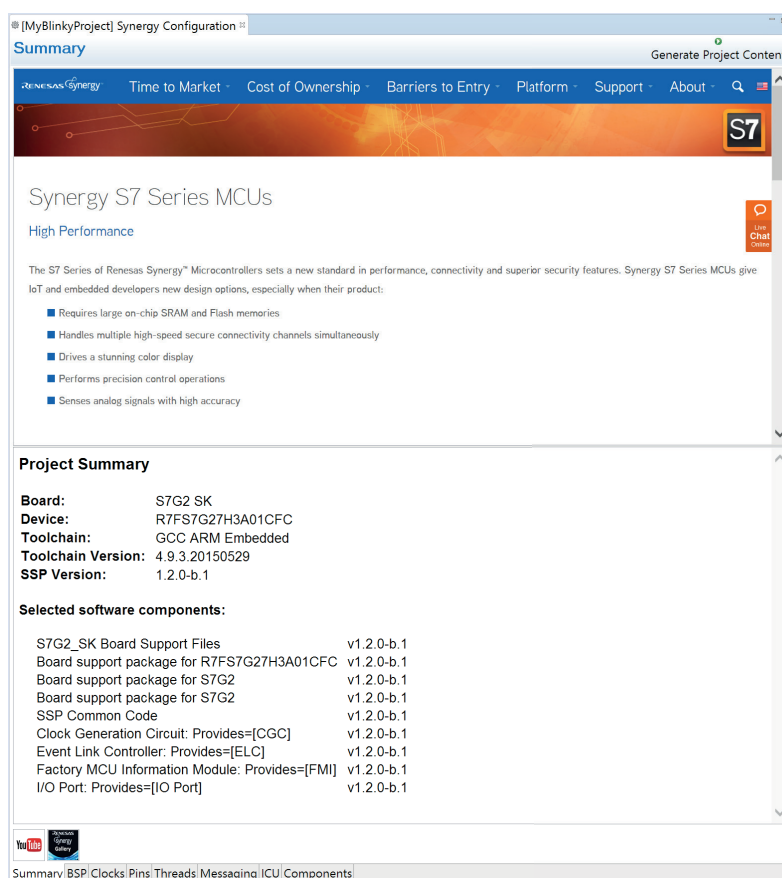
Figure 8-6: The summary tab of the Synergy Configurator

The project also contains several directories with »synergy« in the name, containing source-, include- and configuration files for the SSP. It is a general rule that the contents of these folders (and subfolders) should not be modified. They contain files generated by the configurator and any modification made there will be lost the next time the project contents is generated or refreshed. The user editable source files are those directly in the root of the *\src* folder or any other folder added by you.

Now it is time for you to write the first real Synergy Platform source code. The plan is to alternate between the green LED1 and orange LED3 on the S7G2 Starter Kit every second, so you will have to add code for turning them on and off and for a delay loop. How to do that?

There are actually two options for that: One is using the API of the Hardware Abstraction Layer directly and one is using the HAL together with the BSP. Which one do you think is the better one? You can review chapter 2, if you are unsure about the answer.

Looking at the code in the file *\src\synergy_gen\hal_data.c*, we find that there is the following definition for the I/O port driver instance `g_ioport`:

```
const ioport_instance_t g_ioport =
{ .p_api = &g_ioport_on_ioport, .p_cfg = NULL };
```

`g_ioport_on_ioport` is a structure, which declares the possible actions for the ports and is assigned to the API-pointer of the `g_ioport` instance. The contents of the structure can easily be viewed by hovering with the mouse over it, which will reveal that one of its members, `.pinWrite`, is a pointer to the pin write function.

So turning an LED on, you could write:

```
g_ioport.p_api->pinWrite(ioport_port_pin_t pin,
                         IOPORT_LEVEL_LOW);
```

But this means that you would actually need to know which I/O-ports LED1 and LED3 are connected to and how many LEDs are available for use! For that, we could either read the documentation of the board or scrutinized the schematics to find the correct port. Or simply use the BSP API, which provides the structure `bsp_leds_t` for that purpose. Calling the BSP function:

```
R_BSP_LedsGet(bsp_leds_t * p_leds);
```

will then populate an user defined variable of the type `bsp_leds_t` (e.g. `Leds`) with the necessary values.

You can turn on LED1 by writing:

```
g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                         IOPORT_LEVEL_LOW);
```

You will need a second statement, turning LED3 off, by setting its pin-level to high.

Finally, we need to provide a delay to have the LEDs toggle in a user friendly way. For that, we again call a BSP API:

```
R_BSP_SoftwareDelay(BSP_DELAY_UNITS_SECONDS, 1);
```

This will create a one second delay loop. The macro `BSP_DELAY_UNITS_SECONDS` defines the units for the delay, either in seconds, milliseconds or microseconds, while the 1 stands for the number of units to delay. More information about each function call or variable in the code can be viewed by using the Smart Manual feature of e[2] studio.

All what is needed once this is done, is to copy/paste the three lines of code and to reverse the pin-levels of the LEDs in the second set. And finally, as we want to run the program indefinitely, a while(1)-loop needs to created around the code.

What is left now, is to insert the following lines of code into the *hal_entry.c* file:

```c
void hal_entry(void)
{
    bsp_leds_t Leds;
    R_BSP_LedsGet(&Leds);

    while (1)
    {
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                 IOPORT_LEVEL_HIGH);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                 IOPORT_LEVEL_LOW);

        R_BSP_SoftwareDelay(BSP_DELAY_UNITS_SECONDS, 1);

        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                 IOPORT_LEVEL_LOW);
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED3],
                                 IOPORT_LEVEL_HIGH);

        R_BSP_SoftwareDelay(BSP_DELAY_UNITS_SECONDS, 1);
    }
}
```

While writing the code, you can always use the auto-completion feature of e² studio. Just press *<ctrl>-<space>* and a window displaying possible completions for the structure or function. If you click on an entry, it will be automatically inserted into the code.

It is now your turn: Please enter the lines of code above into the *hal_entry.c* file in your project. For that, expand the *src* folder of your project and double-click on the file. This will open it in the editor. If you do not want to type everything, you can also download a complete project from the Website of this book (www.renesas.com/synergy-book).

8.1.4 Compiling the First Project

When you did all the typing, the program is ready to be built. There are two different configurations for a build: Debug and Release. The Debug configuration will include all information necessary for debugging a program, like variable and function names and will also turn off certain optimizations of the compiler, for example loop unrolling. This makes debugging easier, but will create larger and slower code. The Release configuration will strip all this information from the output file and turn on full optimization, thereby creating smaller and faster code, but you will no longer, for example, be able to view variables, unless you know their address in memory.

For your first test, the Debug configuration, which is also the default, is the way to go. To build your project, click on the build-button on the main menu bar 🔨 and the process will start. If you did everything right, the compilation will finish with 0 errors and 0 warnings. If there are compile-errors, you need to go back to your code and double-check, if you entered everything correctly. If not, change your code accordingly.

After the build succeeds the output file *MyBlinkyProject.elf* has been created, which needs to be downloaded to the processor before we can run and debug it.
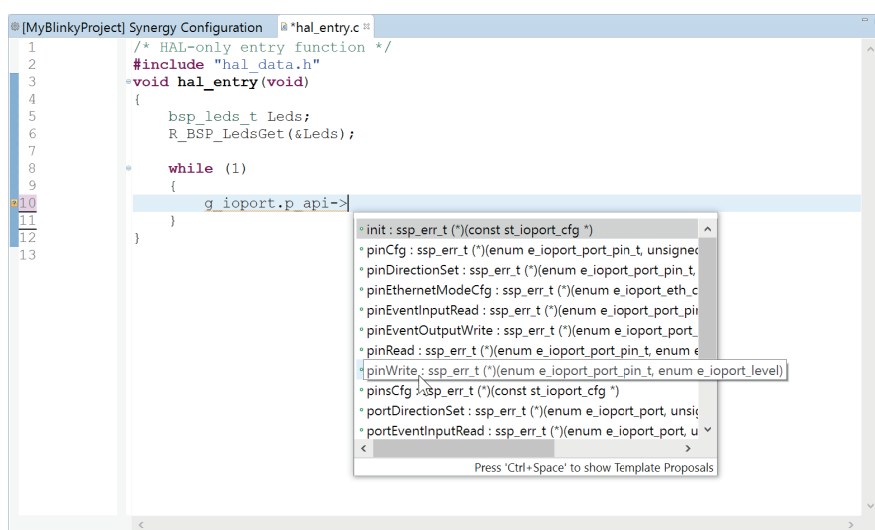
Figure 8-7: Pressing <ctrl>-<space> at a variable or function will activate the code completion feature of the editor

8.1.5   Downloading and Debugging the First Project

The next step will be to actually run the program on the SK. And this is now the right moment to connect the kit to your Windows® workstation: Insert the small plug of the USB-cable delivered with the board into the connector called *DEBUG_USB* and the other end into the PC. The green LED4 at the top right hand corner of the PCB should be lit,indicating that the board has power. If the kit just came out of the box, the pre-programmed demo will run, signalling that everything is working as expected. Windows® might display a dialog indicting the installation of the J-Link® OB Debugger, which should be completed automatically.

DOWNLOADING

To download our program, we will have to create a debug configuration first. Click on the small arrow beside the Debug symbol 🐞 ▾ and select *Debug Configurations* from the drop down list.

In the window showing up, highlight *MyBlinkyProject Debug* under Renesas *GDB Hardware Debugging*. Finally click on *Debug* at the lower right corner of the window. This will launch the debugger, download the code to the Synergy device and will ask you, if you want to switch to the *Debug* perspective. Answer yes. The *Debug* perspective will open and the program counter is set to the entry point of the program, the reset handler. This configuration needs to be done only once. The next time you can start the debugger by only clicking on the Debug symbol.
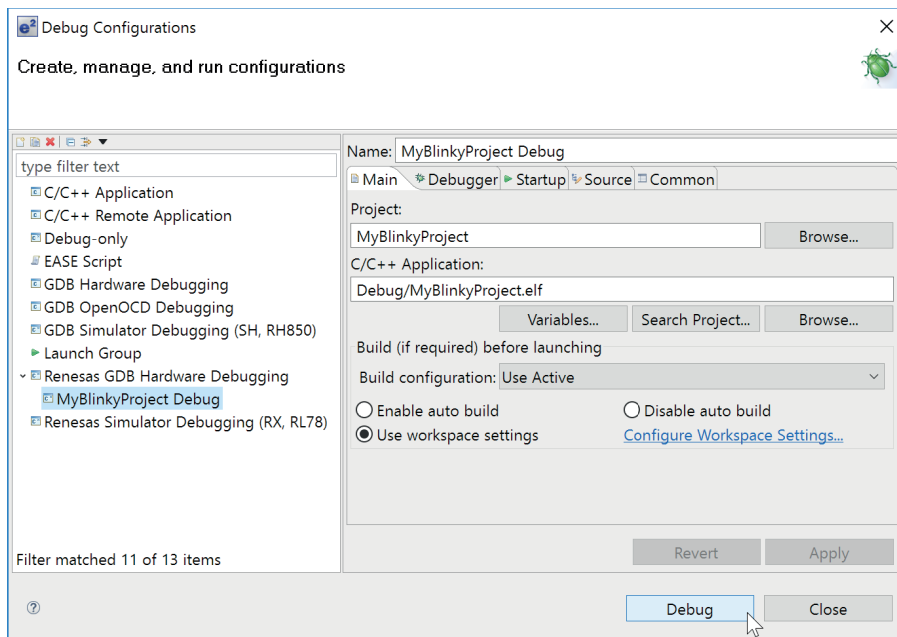
Figure 8-8: Once MyBlinkyProject is selected, no change needs to be made on the different tabs

RUNNING

Click on the Resume button ▶ and the next stop will be at `main()`, at the call to `hal_entry()`. Click again and the program continues to execute, toggling the green and the orange LEDs on the Starter Kit in a one second interval as programmed.

WATCHING THE RESULTS

If everything is working as expected, click on the Suspend button ⏸ on the main menu bar. This will stop the execution of the program without terminating it. In the editor view, activate the tab with the file *hal_entry.c*, and right-click in one of the lines with a write to the ports; in the menu showing, select *Run to line*. Execution will resume and the program will stop at the line you clicked in. Now have a look at the view with the variables tab at the right. You will see the `Leds` structure listed. Expand it and browse and analyse the different fields. This view will come handy when debugging a larger project
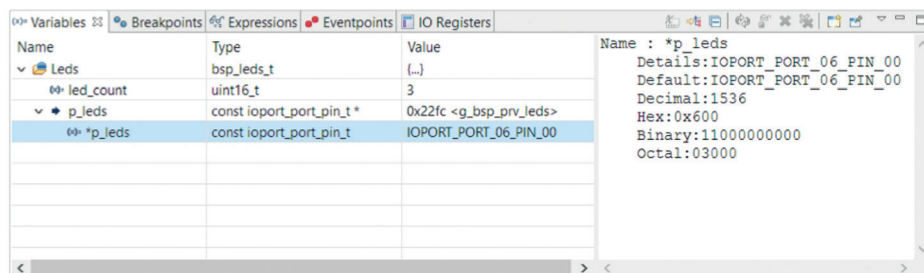
Figure 8-9: Variables and their values can be observed in the variables tab

The final step is to disconnect the debugger from the board by clicking on the disconnect button, ⬛ stopping the execution of the program.

CONGRATULATIONS!

You mastered your first program for the Synergy Platform!

Points to take away from this chapter:

- The Project Configurator creates all files and settings necessary for a new project.
- The Synergy Configurator allows the SSP and the runtime environment to be configured easily based on a graphical user interface.
- A Debug Configuration is necessary for debugging the project, but it is created automatically and just needs to be activated.
- Only very few lines are needed for the implementation of the actual code.

9 INCLUDING A REAL-TIME OPERATING SYSTEM

What you will learn in this chapter:

- What threads, semaphores and queues are, and how to use them.
- How to add threads and semaphores to e² studio.
- How to toggle an LED with a push-button under RTOS control.

The exercise in the previous chapter already made use of a good part of the Renesas Synergy™ Software Package (SSP). In this chapter you will create a small application using the ThreadX® RTOS utilizing a thread for the LED and a semaphore for synchronization with a push-button, and you will experience first-hand that only few steps are actually necessary for this.

You will create the complete project from scratch, so don't worry if you haven't done the previous labs.

9.1 Threads, Semaphores and Queues

Before we dive into actually doing this exercise, let's define some of the terms we will use in this and in the next chapter, to make sure that we all have a common understanding.

First we need to define the term *Thread.* If you are more accustomed to the term *Task*, just think of a thread being a kind of task. Some even use both terms alternately. When using an RTOS, the application running on the microcontroller will be broken down in several smaller, semi-independent chunks of code, with each one typically controlling a single aspect of it. And this small pieces are called *threads*. Multiple threads can exist within one application, but only one can be active at any given time, as Synergy microcontrollers are single core devices. Each one has its own stack space, an assigned priority in respect to the other threads in the application and can be in different states like ready, completed or sleep. In ThreadX®, the control block of a thread contains a member called `tx_thread_state`, from where the application can read the current state. Inter-thread signalling, synchronization or communication is performed by the means of semaphores, queues, mutexes or event flags.

A *semaphore* is a resource of the RTOS, which can be used for signalling events and for thread synchronisation (in a producer-consumer fashion). Using a semaphore allows the application to suspend a thread until the event occurs and the semaphore is posted. Without an RTOS, is would be necessary to constantly poll a flag variable or to create code to perform a certain action inside an interrupt service routine (ISR), blocking other interrupts for quite some time. Using semaphores allows you to exit ISRs quickly and to defer the code to the associated thread.

ThreadX® provides 32-bit counting semaphores, and each semaphore has two basic operations associated: `tx_semaphore_get`, which will decrease the semaphore by one, and `tx_semaphore_set`, which increases the semaphore by one. Semaphores in ThreadX® are public resources. It's worth noting that sempahores, like all ThreadX® objects, cannot invoke suspension outside of a thread, for example inside the interrupt service routine. Therefore, all related functions must be called with `TX_NO_WAIT` only when called from an ISR.

The last term we need to talk about, even if we do not use one in this exercise, is the *queue*. But we will do so in the next chapter. Message queues are the primary method of inter-thread communication. One or more messages can reside inside a message queue. There is a special case: A queue with a single message is called *mailbox*. Allowed message sizes are 1 through 16 32-bit words. Larger messages need to be passed by a pointer, something implemented already in the SSP Messaging Framework (`sf_message`) and the size needs to be specified when the queue is created. There is no limit on the number of queues in ThreadX®. Messages are put into the queue using the `tx_queue_send()` function and read from the queue by `tx_queue_receive()`. New messages are placed at the end of a queue and received messages are removed from the front.

9.2   Adding a Thread to ThreadX® using e² studio

This exercise is again based on the Synergy Starter Kit SK-S7G2. This time, we will use the push-button SW4 on the top right corner of the board to signal an event to the application, which will toggle the green LED1 as response to it. For the implementation, we will use ThreadX® and the handling of the event will take place inside a thread and the notification of this thread will be done through a semaphore.

As usual, the first step is to create a new project using the Project Configurator, something you already exercised in chapters 4 and 8. To get started, go to *File → New → Synergy C Project*, enter a project name, for example *MyRtosProject*, verify that the settings are OK and select the *S7G2 SK* on the next screen. On the *Project Template Selection screen*, select BSP.

Click on *Finish* and after the project has been generated by the configurator, e² studio will switch to the *Synergy Configuration* perspective. Go directly to the *Threads* tab. This tab will show one entry for *HAL/Common* in the *Threads* pane, containing the drivers for the Event Link Controller (ELC), the I/O port driver and the Clock Generation Controller (CGC) and the Factory MCU Information (FMI). The FMI-driver includes a generic API for reading the records from the Factory MCU Information Flash Table, providing information about the features and peripherals implemented on the microcontroller used. Click on the *New Thread* icon at the top of the pane, which will add a new thread.
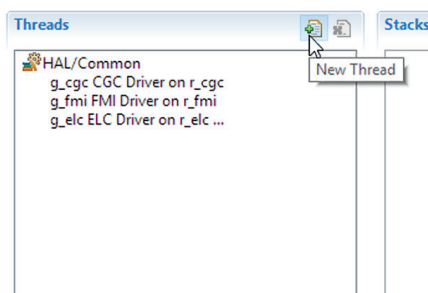

Figure 9-1: After the Synergy Configurator starts, only one thread will be shown. Clicking on the "Add Thread" symbol will add another one

Now change the properties of the new thread: Rename the symbol to *led_thread* and the name to *LED Thread*. Leave the other properties at their default value. On the *LED Thread Stack* pane, click on the New icon and select *Driver → Input → External IRQ Driver on r_icu* (see Figure 9-2).
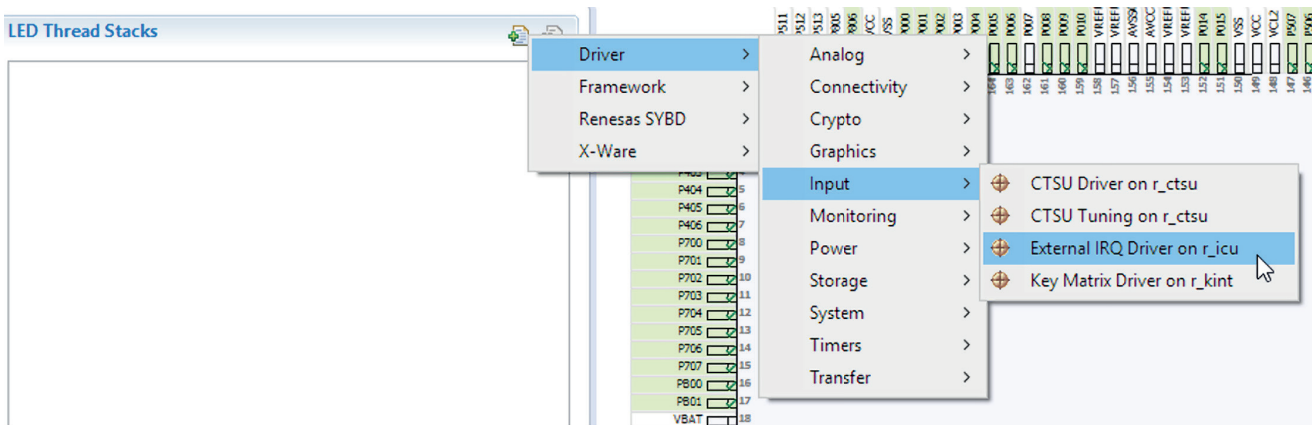
Figure 9-2: Adding a new driver takes only a few mouseclicks

This will add the driver for the external interrupt, but it shows up in red, indicating that there are additional changes to make. Hover with the mouse over it and look for what needs to be done (Figure 9-3).

The interrupt is enabled by changing the *Interrupt Priority* from *Disabled* to *Priority 8* in the Properties window. Actually, it could be any other priority between 0 and 14 as well, but 8 is a good start, as you will rarely run into interrupt priority collisions, even in larger systems. Please note that priority 15 is reserved for System Tick Timer *(systick)* and hence cannot be used by other peripherals.

While you are at the properties for *g_external_irq0*, make some other changes as well. First, change the *Channel* from 0 to 11, as SW4 is connected to *IRQ11*. For the same reason, change the name to *g_external_irq11*, or anything else you like.
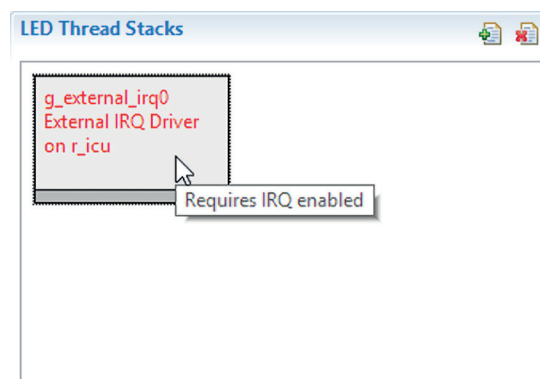
Figure 9-3: Hovering with the mouse will bring up an explanation of which property needs to be changed

Change the *trigger* from Rising to Falling, to catch the pressing of the button and the Digital *Filtering* from *Disabled* to *Enabled*. This will help to debounce the button. And finally, change the *Callback* from *NULL* to *external_irq11_callback*. This step will create a function, which is called once SW4 is pressed. We will add the code for the callback function itself later on. Figure 9-4 shows a summary of the necessary settings.

Property	Value
∨ Common	
Parameter Checking	Default (BSP)
∨ Module g_external_irq11 External IRQ Driver on r_icu	
Name	g_external_irq11
Channel	11
Trigger	Falling
Digital Filtering	Enabled
Digital Filtering Sample Clock (Only valid when Di	PCLK / 64
Interrupt enabled after initialization	True
Callback	external_irq11_callback
Interrupt Priority	Priority 8 (CM4: valid, CM0+: invalid)

Figure 9-4: The properties of the IRQ driver needed for our application

Now, there are only a couple of additional steps left you need to perform until you can compile and download your program. The next one is to add a semaphore.

For that, click on the *New Object* button in the *LED Thread Objects* pane. If you do not see this pane, but a *HAL/Common Objects* pane, highlight the *LED Thread* in the *Threads* pane and it will become visible. Add a *Semaphore*, as we will need one to notify the *LED Thread* once the button is pushed. Change the semaphore's *Name* property to SW4 Semaphore and the *Symbol* property to g_sw4_semaphore.

Leave the count at zero, as we will increment it each time the button SW4 is pressed. Now your *Threads* tab in the Synergy Configurator should look similar to Figure 9-5.
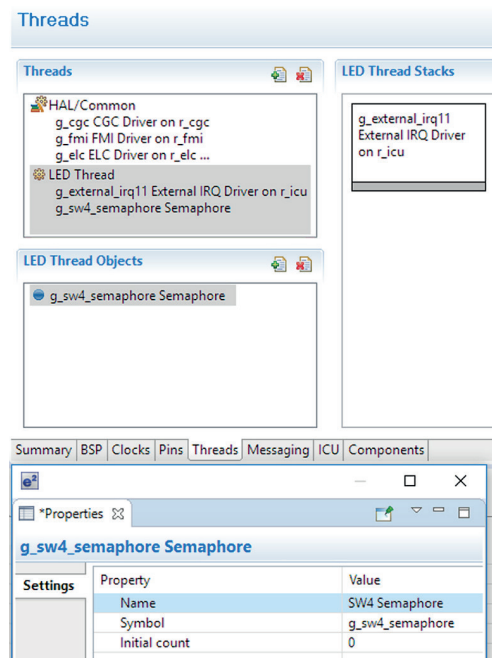
Figure 9-5: This is what the Threads tab should look like after the LED Thread and the semaphore has been added

The final step in the Synergy Configurator is to configure the I/O-pin to which SW4 is connected to as IRQ11 input. For that, activate the *Pins* tab inside the configurator and expand *Ports → P0* and highlight *P006*. On the S7G2 Starter Kit, this is the pin SW4 is connected to. In the *Pin Configuration* pane at the right give it the *Symbolic* Name SW4, change the mode to *Input mode* and the IRQ to *IRQ11_DS* (if not already set) and the *Chip input/output* to GPIO. Note that the package viewer at the right will highlight pin 163 / P006. The complete configuration is shown in Figure 9-6.
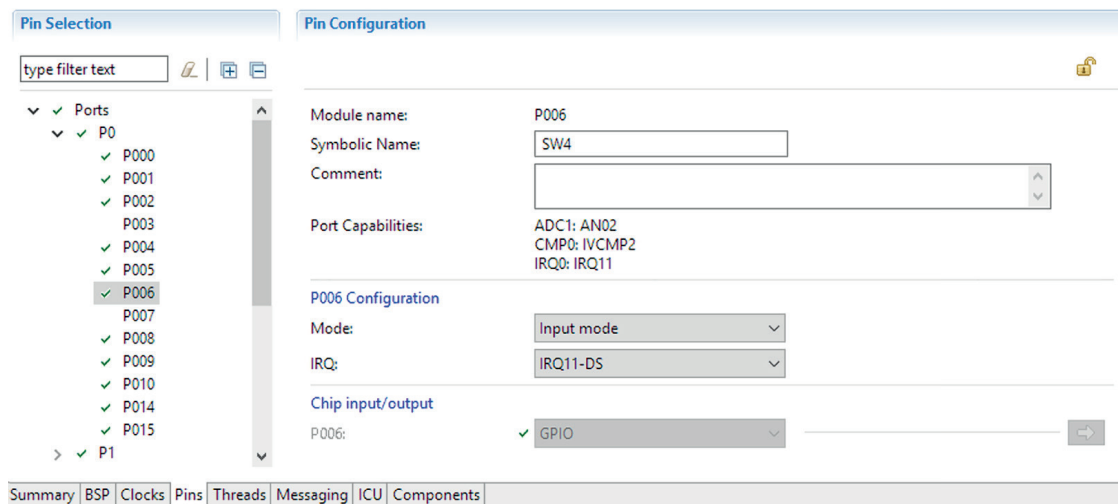
Figure 9-6: Port P006 needs to be configured as input to IRQ11

With this done, you have finished the settings in the configurator. Save the changes and click on the *Generate Project Content* icon at the top of it to generated the necessary files, folders and settings.

The final task you need to execute is to add code for populating the `Leds` structure similar to what you did already in the exercise of chapter 8, write a couple of lines for toggling the LED and to read the semaphore and to create the callback function. The full code for this can be reviewed below.

As we are using a thread for the handling of the push-button and toggling of the LED, we need to add the related code to the *led_thread_entry.c* file this time. Double-click on the file in the *Project Explorer* to open it in the editor. If it not visible, expand the project folder and then the `src` directory. As with the exercise in chapter 8, you will need to add the structure for the LEDs and initialize it. Another variable for the level of the I/O-pin the LED1 is connected to needs to be defined. Name it `led_level`. It should be of type `ioport_level_t` and should be initialized to `IOPORT_LEVEL_HIGH` (a "high" level corresponds to "off" on the SK-S7G2).

The next step will be to open and to configure the external IRQ pin connected to SW4 on the board. For this, use the *open* function of the IRQ HAL driver. With that, the initialization steps necessary are finished.

Inside the `while(1)` loop, you need to add a couple of statements and to remove the `tx_thread_sleep(1)` statement. Start with a function call to write the value of the `led_level` to the output register of the I/O-pin for LED1 followed by statements toggling the pin level. There are several ways to do that. Implement your own, or look up the code at the end of the chapter. Do not forget the Smart Manual feature of e[2] studio, it helps a lot!

The final statement inside the `while(1)` is a call to `tx_semaphore_get()` with the address of the semaphore and the constant TX_WAIT_FOREVER as parameters. This will advise the RTOS to suspend the thread indefinitely until the semaphore is posted from the callback functioninside the IRQ 11 interrupt service routine.

And, the last thing to do is to add the callback routine called by the IRQ 11 ISR. This code should be as short as possible, as it will be executed in the context of the interrupt service routine. You can find the prototype for this function in the *HAL/Thread* code generated in the *synergy_gen* folder. In our case, it can be copied over from the *led_thread.h* file residing there:

```
void external_irq11_callback(external_irq_callback_args_t *p_args);
```

Inside the callback function, add the following line of code to set the semaphore once the button SW4 is pressed:

```
tx_semaphore_put(&g_sw4_semaphore);
```

Once all the coding is complete, build your project by clicking on the *Build*-icon (the "hammer"). If any warning appears, please ignore. If it does not compile with zero errors, go back to your code and fix the problems with the help from the feedback of the compiler looking them up in the *Problems* view.

If the project built successfully, click on the small arrow beside the *Debug* icon, select *Debug Configurations* and expand *Renesas GDB Hardware Debugging*. Select *MyRtosProject* or the name you gave your version and click on *Debug*. This will start the debugger. If you need more information on that, please review the related section in . Once the debugger is up and running, click on *Resume* twice. Your program is now running and each time you press SW4 on the SK, the green LED1 should toggle.

```
#include "led_thread.h"
void led_thread_entry(void);
bsp_leds_t Leds;
/* LED Thread entry function */
void led_thread_entry(void)
{
    ioport_level_t led_level = IOPORT_LEVEL_HIGH;
    R_BSP_LedsGet(&Leds);
    g_external_irq11.p_api->open(g_external_irq11.p_ctrl,
                                 g_external_irq11.p_cfg);
    while (1)
    {
        g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1],
                                 led_level);
        if (led_level == IOPORT_LEVEL_HIGH)
        {
            led_level = IOPORT_LEVEL_LOW;
        }
        else
        {
            led_level = IOPORT_LEVEL_HIGH;
        }
        tx_semaphore_get(&g_sw4_semaphore, TX_WAIT_FOREVER);
    }
}
void external_irq11_callback(external_irq_callback_args_t * p_args)
{
    tx_semaphore_put(&g_sw4_semaphore);
}
```

CONGRATULATIONS!

You successfully finished this exercise.

Points to take away from this chapter:

- Using HAL-driver is simple through the use of comprehensive APIs.
- The SSP will take care of most non-user code related things.
- Using ThreadX® is straightforward and adding threads and semaphores is not much of an effort.

10 SENDING DATA THROUGH USB USING A QUEUE

> **What you will learn in this chapter:**
>
> - How to setup an USB-transfer using one of the Renesas Synergy™ Application Frameworks and how to receive the data on a host workstation.

In this chapter, we will use the communication framework inside the Application Frameworks to send the state of LED1 through an USB-port to the Windows® work-station. For this, you will to add a new thread and a queue to your project from the previous chapter 9 and modify the existing LED thread to send the state as a string to the queue. The new communications thread will then send the string to a terminal program on the host using USB. While programming this exercise, you will experience once again the simplicity the Synergy Software Package (SSP) provides to users, even when setting up complex communications like USB.

If you haven't done the exercise from chapter 9, you can download the project from the book's website (www.renesas.com/synergy-book). Import it according to the instructions given in chapter 5.1.3 and you are all set. Of course, you can download a full project for this chapter as well, if you just want to see how the goal is achieved. If you are using IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy), you can also download a complete project, if you do not want to follow the instructions below and adjust them as necessary.

10.1   Setting Up an USB Port in Synergy Platform

If you closed e² studio after the last exercise, re-open it and make sure that your project *MyRtosProject* is active. If not, just click on it and it will be displayed in bold. The first step for you is to switch to the *Synergy Configuration* perspective and to go to the *Synergy Configuration [MyRtosProject]* view. If this view was closed before, you can re-open it by double clicking on the *configuration.xml* file in the *Project Explorer* view, or by clicking on the *Gear* icon on the main menu bar. As you are a Synergy expert by now, I will not describe every step in detail, if it was already detailed in a previous exercise.

Select the *Threads* tab and add a new thread with the symbol *comms_thread* and the name Comms Thread. With the newly added thread selected, add a communications framework instance to it. Click on the *Add Stack* icon in the *Comms Thread Stack* pane and select *New → Framework → Connectivity → Communications Framework on sf_el_ux_comms*.
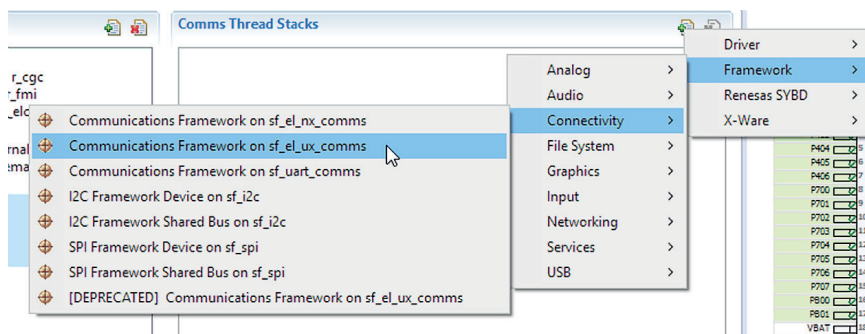

Figure 10-1: First step is to add the communications framework

This will add the complete stack of the *g_sf_comms0 Communications Framework on sf_el_ux_comms* to the system, down to a level where an user intervention is necessary. You might wonder what the meaning of different color bars of the thread modules is. It is quite simple: Regular instances are marked in grey, common instances in blue (there is just one global per project) and pink marks options.

For our project, one USBX port is needed. To add it, click on the optional Add USBX Port DCD module and select New and USBX Port DCD on sf_el_ux for USBFS. In the Properties for this module, change the Full Speed Interrupt Priority property to e.g. Priority 8 (see Figure 10-2).
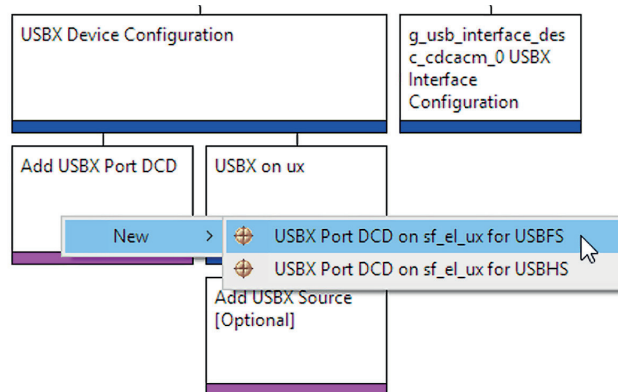
Figure 10-2: An USB Full-Speed port needs to be added to the system

You might also want to have a look at the *USBX Pool Memory Size* in the properties of the *USBX on ux* module. It default to 18 kBytes, which is sufficient for most tasks. If you want to use different transfer classes in the future, you have to adjust the pool size accordingly. Table 20 of the USBX documentation in the file *_express_logic_u_s_b_x.html*, which was copied to your computer during the installation of the SSP, has all the details for that.

With the *Comms Thread* still selected, add the queue we need for sending the data in the *Comms Thread Objects* pane. Name it *CDC Queue* and assign it the symbol `g_cdc_queue` in the Properties view. Also set the *Message Size* to 3, as we want to transmit 3 words with 4 bytes each in each transfer and the *Queue Size* to 24 bytes, meaning that the queue has space for two (12-byte) messages.

With this, the configuration of the SSP is complete. Save the configuration and click on the *Generate Project Content* button. Note that the file *comms_thread_entry.c* has been added to the *src*-folder of your project. You will add code to this file during the next steps.
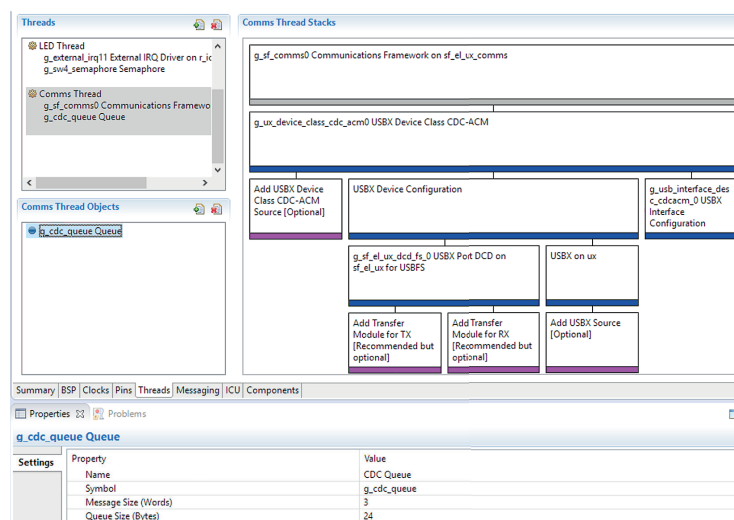
Figure 10-3: With all additions and modifications made, the Threads tab should look like this

But before you add the code needed to transfer the state of the LED to the host work-station, you will have to add code to the LED Thread to actually copy the message into the queue. For this, switch back to the *C/C++ Perspective* and open the file *led_thread_entry.c*. At the top, add an include directive for the file *comms_thread.h* to share elements, like the `g_cdc_queue`, from the Comms Thread with the code in the LED Thread. As second step, add a global array named `send_str` with 24 characters. Remember, this is the size of our queue. And finally, add the following lines just after the ioport write statement inside the `while(1)` loop in this file:

```
g_ioport.p_api->pinWrite(Leds.p_leds[BSP_LED_LED1], led_level);

if(led_level == IOPORT_LEVEL_HIGH)
{
    strcpy(send_str, "LED off\n\r");
    led_level = IOPORT_LEVEL_LOW;
}
else
{
    strcpy(send_str, "LED on\n\r");
    led_level = IOPORT_LEVEL_LOW;
}

/* Send the message in the queue. Wait forever for space */
/* to be available in the queue for the message.         */
tx_queue_send(&g_cdc_queue, send_str, TX_WAIT_FOREVER);

tx_semaphore_get(&g_sw4_semaphore, TX_WAIT_FOREVER);
```

The final step is now to add code to the *comms_thread_entry.c* file. Open it from the *Project Explorer*.

Edit the *comms_thread_entry.c* file and add a global array of 24 elements of the type uint8_t named `rx_msg`. After that, replace the line `tx_thread_sleep(1);` inside the `while(1)` loop with the following:

```
tx_queue_receive(&g_cdc_queue, rx_msg, TX_WAIT_FOREVER);

g_sf_comms0.p_api->write(g_sf_comms0.p_ctrl,
                          rx_msg,
                          strlen(rx_msg),
                          TX_WAIT_FOREVER);
```

This concludes all the coding necessary. You may ask now: "Wait a minute: Two lines of code is all what's needed to transfer a string received by a queue over USB? Isn't that too easy?" And the answer is: No, it is so easy. The SSP and the Application Frameworks will take care about everything else. Isn't that really simple. Remember the last time you coded a transfer on another platform? I would guess, there was a huge difference in the effort needed for that.

Now, all what is left, is to build the project. The first time you do that, it will take some time, as the code for the communications framework will need to be compiled as well. Once the project is built with zero errors (you might get some warnings about unused parameters; just ignore them), start the debug session and with the *Debug* perspective open, click on *Resume* twice to start the program. As a first test, press SW4 once to see if LED1 still toggles.

10.3   Setting Up a Receiver On the Host Side

With the program running, connect a second USB type A to micro B cable to the USB port labelled J5 at the bottom of the Starter Kit. Insert the other end into your Windows® workstation and wait a couple of seconds until Windows® recognizes the board and installs the drivers for it.

Start a terminal emulator program. During the development of this exercise, we used Tera Term Pro, which can be downloaded from https://ttssh2.osdn.jp/ and found it quite useful. In Tera Term, you will see the CDC serial port listed. In Figure 10-3 it is COM6, but it is likely to be different on other workstations. If you are not sure, use the Device Manager of Windows® to find out the port the board is connected to.
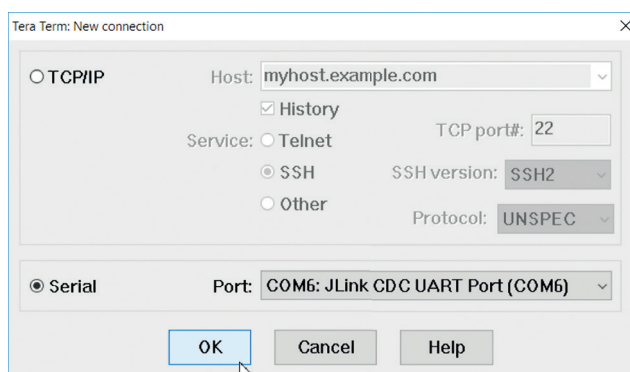
Figure 10-3: If Windows® recognized the board correctly, it will be listed in Tera Term as serial connection

In case, the board is not listed at all or the Device Manager indicates an error, there might be a problem with the driver. Please refer to the following article on the Renesas Synergy™ Knowledge Base to resolve that problem:
https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy™_Platform/Renesas_Synergy_Knowledge_Base/Installing_USB_CDC_Driver_on_Windows_10.

With the connection made and Tera Term running, press SW4 a couple of times and you should see the LED1 toggling and the state of it output to the terminal as shown in Figure 10-4.
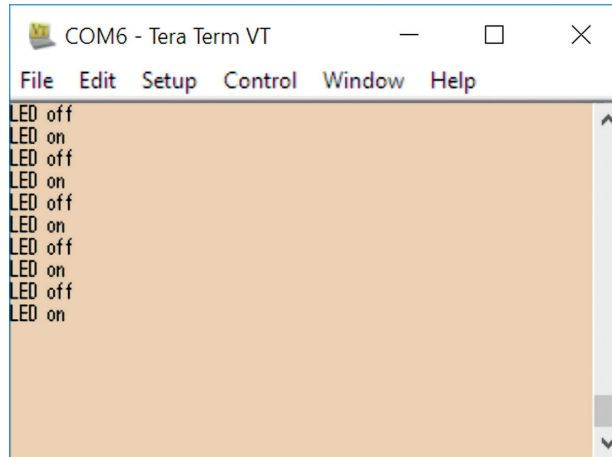

Figure 10-4: With the transfer running, the terminal program will display the state of LED1 each time SW4 is pressed

CONGRATULATIONS!

You just finished this exercise. And what do you think now: Was this easy or not?

Points to take away from this chapter:

- Adding an USB transfer is easy if the Application Frameworks are used.
- Only very few lines are needed for the implementation.

11 EVENT ANALYSIS WITH TRACEX®

What you will learn in this chapter:

- What TraceX® is and how it can help you debugging your code.
- How to install and use TraceX® and how to interpret the data collected.

Tracing events at run-time is an important debug capability, as it provides visibility into the sequence of Real-Time Operating System (RTOS) activities of an application.

In this chapter, we will employ TraceX®, a Windows® workstation-based tool to trace all the events happening when we press the push-button SW4 on the Renesas Synergy™ Starter Kit (SK) in the exercise from chapter 10. For this we will modify the program we used there. If you have not done the hands-on of chapter 10, this is not a problem. You can download the complete project from the website for the book (www.renesas.com/synergy-book). Once downloaded, you only need to import it according to the instructions given in chapter 5.1.3.

11.1 An Introduction to TraceX®

TraceX® is host-based and integrates seamlessly with the Synergy Software Package (SSP). It provides developers with a graphical view of real-time events, enabling them to better understand the behaviour of their system. Visualized are occurrence and sequence of system events like interrupts, context switches or the setting of a semaphore, events which are normally out of the view of standard debugging tools. This way, problems can be easily diagnosed and the system fine-tuned to optimize performance and efficiency.

The trace information itself is stored in a buffer on the target system, with the buffer location and size determined in the Properties of the TraceX® module during configuration. TraceX® implements a circular buffer, which enables the most recent "N" events to be available for inspection, even in the event of system malfunction or other significant events

Other features and benefits of TraceX® include:

- Full integration with ThreadX® and all other X-Ware™ components, but it runs independently on the host postmortem or at break-points
- Detect priority inversions
- Task execution profiles
- Stack usage
- Delta ticks between events
- Performance analysis and bottleneck elimination
- NetX™, and FileX® statistics
- Raw Trace Dumps and Trace Buffers

All these features save a lot of time during the development of an application. There is no need to instrument your program with printf()-statements or with toggling LEDs to visualize a sequence of events in question if you need to isolate a certain problem in your code. This really helps you to save time and energy!

11.2   Using TraceX® with e2 studio

In this part of the chapter we will only cover the usage of TraceX® together with e^2 studio. If you want to use the IAR Embedded Workbench® for Renesas Synergy™ (EW for Synergy), please download the web-version of this chapter.

11.2.1   Installation

TraceX® is not included in the standard e^2 studio installation and needs to be downloaded separately from the Synergy Gallery. On the Synergy Gallery, select *Development Tools* and then *TraceX*. The homepage for TraceX® will show and clicking on the blue *Download* button at the top will initiate the download. Read and accept the license agreement appearing and the download of the archive of the installer named *TraceX_for_Renesas_Synergy.zip* will start.

While the installer downloads, please take time to review the information under the *Documentation* tab. The installation instructions given there are very important and you should follow them closely. They also provide you with information necessary, should the automatic installation of the license file fail. This information can later on also be reviewed by opening the file *TraceX-readme.txt*, which is placed in the installation folder of TraceX® after uncompressing the archive.

Once the download is complete, extract the files in the downloaded archive to a new folder of your choice. To start the setup-process right-click on *setup.exe* and select *Run as administrator*.
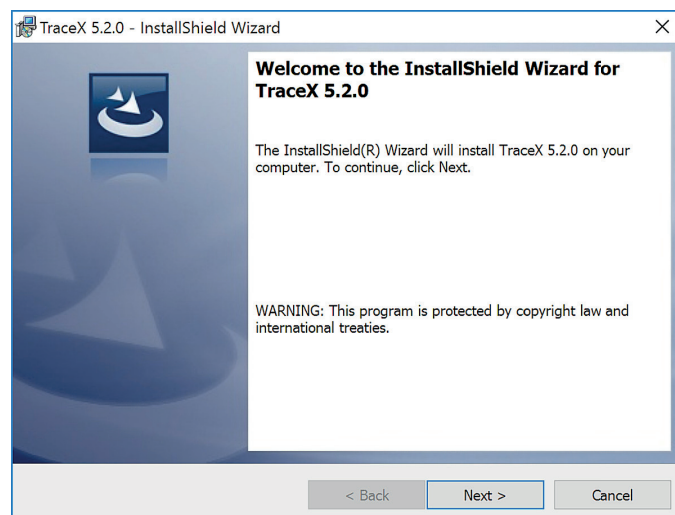
Figure 11-1: The installer for TraceX®

Acknowledge the license agreement for the software once it shows. Follow the instructions of the installer and later on either accept the default path for the installation or change it to a location of your choice. If you do so, please note the new location, as you will need it later to register the tool inside e^2 studio. At the screen showing the summary, click on *Install* and the installation will start.

Once the installation has finished, the last screen has a check box *Launch the program*. It is selected by default, we will do that later from inside the ISDE. For the time being, please deselect the check box.

11.2.2   Setting Up TraceX®

As TraceX® is an external program to e² studio, we need to tell the ISDE where the executable can be found, so that we can run it without leaving the development environment.

Inside e² studio, go to *Window → Preferences* and expand first the *C/C++* and then the *Renesas* branch. Select *TraceX* and navigate to or enter the correct location where you installed the TraceX® executable. If you installed it to the default location, this will be *C:\Express_Logic\TraceX_5.2.0\TraceX.exe*. Click on *Apply* and *OK*.

The next step is to configure your application to perform the actual collection of trace events. To enable tracing, we will need to add the ThreadX® and USBX™ sources.
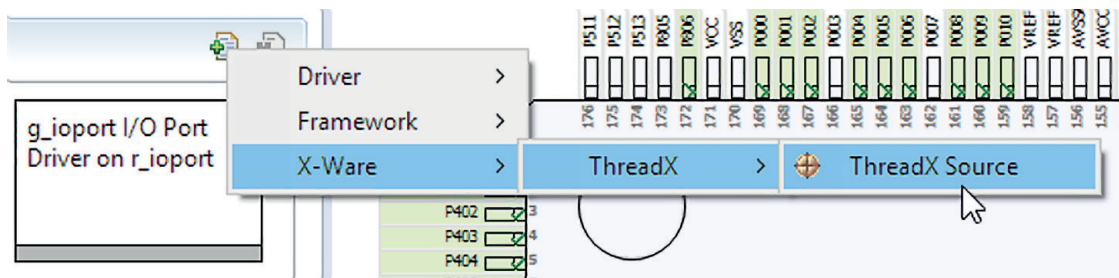
Figure 11-2: The ThreadX® source code needs to be added to the project in order to use TraceX®

For this, switch to the *Synergy Configuration* perspective in e² studio and make sure your RTOS project from the last chapter is active. Switch to the *Threads* tab and select *HAL/Common Modules* in the *Threads* pane. Add the ThreadX® source by clicking on *New → X-Ware → ThreadX → ThreadX Source* in the *HAL/Common Stacks* pane (see Figure 11-2). Select the newly added *ThreadX Source*, and in the *Properties view*, scroll down to *Event Trace* and change it from *Disabled* to *Enabled*. Note the name of the trace buffer (`g_tx_trace_buffer`) and the buffer size (65536). You will need this information later on.
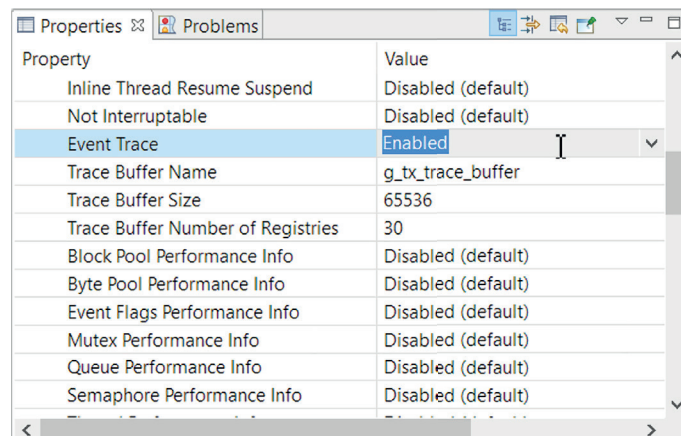
Figure 11-3: The event trace needs to be enabled in the properties of the ThreadX®

Now add the USBX™ source. Select the *Comms Thread* in the *Threads* pane and right-click on the *USBX on ux* module with the pink bar. Select *New →USBX Source*.
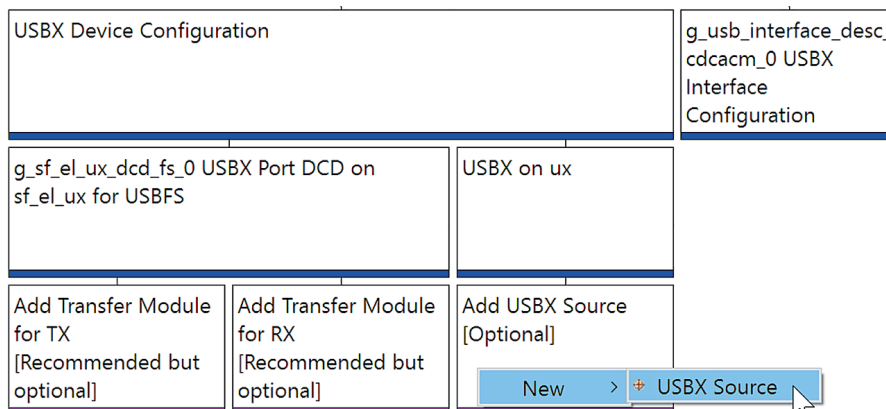
Figure 11-4: For our example, also the source code for USBX™ needs to be added

No other changes need to be made, so save the updated SSP configuration and click on the *Generate Project Content* button to generate the updated versions of the files. Once the generator completes, build the project by clicking on the *hammer* button. Please note that a lot of files need to be compiled this time, so the build will take quite a while. On my workstation, the complete process took a little more than two minutes.

11.2.3   Built-in Views and How to Use Them

TraceX® provides several means to view and to analyse the data. The buttons on tool bar allow you to open files with previously recorded and saved event and to navigate through the different captured events based on a setting in a drop down list (event, object, switch or ID), as well as to generate different statistics.

The large visualization window can display events either in *Sequential View Mode*, which is the default and shows events immediately after each other, no matter how much time has elapsed between them, or in *Time View Mode*, where events are shown in a time relative manner. Switching between these view modes is done by selecting the respective tab at the top of the view window. Both views have their own value when analysing the run-time performance of an application, but in most cases using both of them together will get you closer to your solution. The *Sequential View Mode* lets you quickly assess if the events occurred in the right order and the *Time View Mode* if they took place at the right time.

All events are displayed in the context they happened, which can be seen in the left pane of the window in the *System Context*. Hovering with the mouse over a specific context will display more information. On the event line of a specific context, the different events are shown in individual colours with distinctive abbreviations.

More details about the different displays can be found in the documentation of TraceX®, which can be accessed from the TraceX® download page on the Synergy Gallery. Some of them we will explain in chapter 11.2.4 as well.

11.2.4   Viewing and Interpreting the Data

With the build process complete, start the debugger, but make sure that your Starter Kit is still connected to your workstation with two USB cables, using DEBUG-USB and J5 sockets. e^2 studio will download the code and switch to the debug perspective. Start the execution of the program, by clicking on *Resume* twice. Start your terminal program, but do that not before the application is running, as Windows® or the terminal program might not be able to detect the connection properly if started too early.

With the program running, press the push-button SW4 a couple of times to create sufficient system events. All of them will be recorded in the trace buffer, which was created when setting up the frameworks. Pause the program by selecting *Run → Suspend*, or by clicking on the pause symbol in the main menu bar.

The next step is to launch TraceX® by selecting *Run → TraceX → Launch TraceX Debugging.* In the dialog box showing, the start address of the trace buffer, as well as its size and the path to TraceX® should be pre-populated. If not, please enter the start address of the trace buffer and the size you noted in chapter 11.2.3. Click on *OK*.

If TraceX® does not start, but complains about a missing license file, click on *OK*. TraceX® will then exit. Go to the folder where you uncompressed the installation of TraceX® to and copy the file named *Trace.tag* into the installation folder of the program. This will be c:\Express_Logic\TraceX_5.2.0\, if you installed it into the default location. Go back to e^2 studio (there is no need to close e^2 studio for that procedure) and run TraceX® again. It should now start and show *Synergy_Use_Only* as serial number. This procedure is also explained in the file *TraceX-readme.txt* located in the folder of the installer.

Once the TraceX® window shows, you will see the different events recorded in *Sequential View*. To inspect the ones related to our LED and Comms Thread, move the horizontal slider to the right until you see the events similar to the ones in Figure 11-5.

Context changes are represented by the vertical black lines connecting the context lines. The currently selected event is represented by a solid red line. In the example shown in Figure 11-5, this is event 575. Hovering with the mouse over the different events will give you additional information about them.

TraceX® will display the status of a thread. A green horizontal line indicates that a thread is ready, a purple one that the thread is suspended. After start-up, you will only see the ready status. To see all others, you will need to go to *Options → Status Lines* and select *All On*.
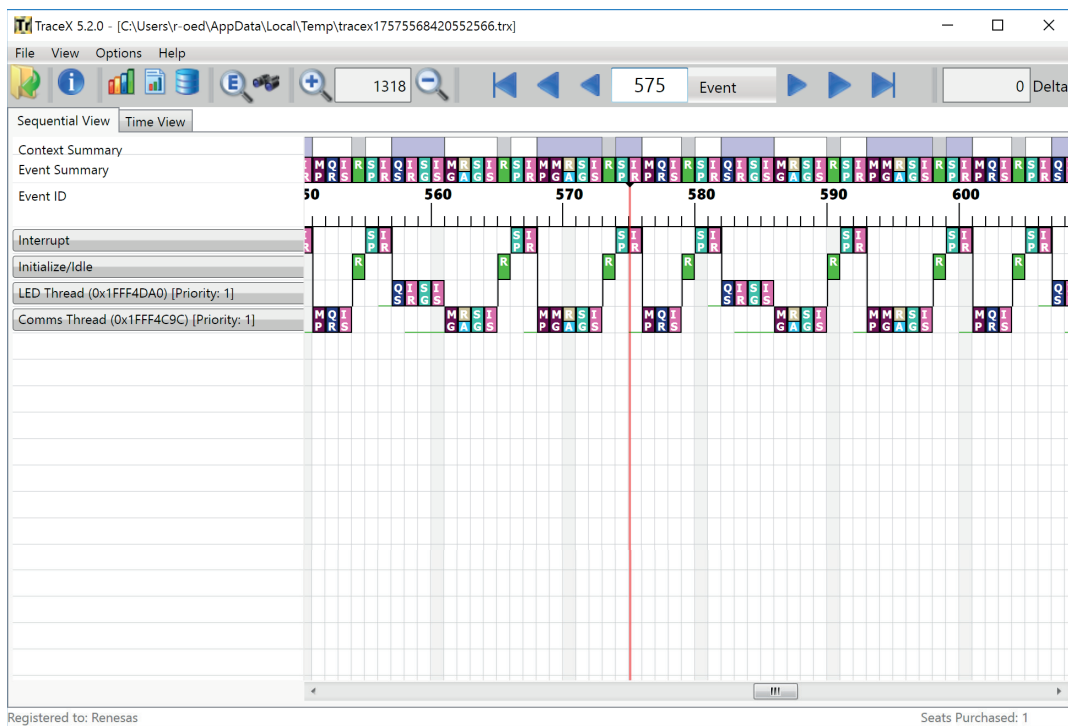
Figure 11-5: The sequence of events as visualized by TraceX®

Each event is represented by a specific colour and abbreviation. In our example, these would be the following (starting at event 579 in Figure 11-5):

- **R:** Running
- **SP:** Semaphore put (SW4 semaphore)
- **IR:** Internal thread resume
- **QS:** Queue send (CDC queue)
- **IR:** Internal thread resume (LED thread)
- **SG:** Semaphore get (SW4 semaphore)
- **IS:** Internal thread suspend (LED thread)
- **MG:** tx_mutex_get (Comms thread)
- **RA:** Device Stack Transfer All Request Abort (USB)
- **SG:** Semaphore get (SW4 semaphore)
- **IS:** Internal thread suspend (Comms thread)

If you want to see detailed information about an event, simply click on one and the *Event Details* will be displayed. A complete list of the possible events and icons can be found inside TraceX® by going to *View* and selecting one of the legends (for example *ThreadX Legend*) or in the User's Guide, which can be accessed through the round blue icon with the white 'i' on the main menu bar of TraceX®. The Users' Guide will also explain all the other features not covered in this short tutorial, like *Time View* (see Figure 11-6) or the different possibilities for performance analysis and statistics.

If you want to collect further TraceX® data, resume the execution of your code in e² studio. Suspend the execution again once enough events have been recorded. Then click on *Run → TraceX* and select *Update TraceX Data*. All that is left in this exercise, is to terminate the debugger by clicking on the Terminate icon on the main menu bar.
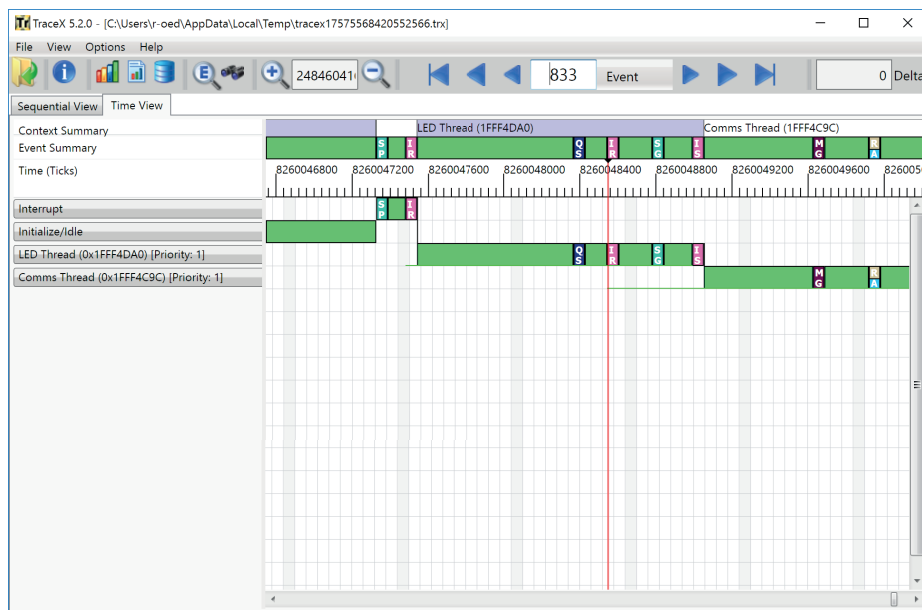
Figure 11-6: When in Time View, TraceX® displays the exact timings between the different events

CONGRATULATIONS!

You just finished the last exercise in this book!

11.3   Using TraceX® With IAR Workbench® for Renesas Synergy™

The printed version of this book mainly covers the toolchain provided by Renesas. If you are using IAR EW for Synergy, you can find an extended version of this chapter including the walk-through for this development environment on the website of the book (www.renesas.com/synergy-book).

Points to take away from this chapter:

- Being able to collect and analyse trace data makes TraceX® a valuable tool in your toolbox.
- Instrumenting a project for TraceX is straightforward and needs only a couple of clicks with the mouse.

12 WHERE TO GO FROM HERE

What you will learn in this chapter:

- What Renesas Synergy™ Solutions' Product Examples (PEs) and Application Examples (AEs) are and how they can help you.
- Which additional software packages and frameworks for the Renesas Synergy™ Platform are availabe.

Now that you have learned what the Renesas Synergy™ Platform consists of and how useful the different parts are for a speedy development, you might ask "Is there more"? The answer is yes! In the past chapters we covered four of the five main elements of Synergy:

- **Synergy Software**
- **Synergy Microcontrollers**
- **Synergy Gallery**
- **Synergy Tools and Kits**
- **Synergy Solutions**

Now in this chapter we will provide details about the fifth element, Synergy Solutions and what they are. In addition we will give a quick insight into Qualified Software Add-Ons (QSAs) and Verified Software Add-Ons (VSAs), the Partner Showcase and additional add-ons provided by Renesas. These solutions go beyond what typical embedded product development kits offer. They are actual products examples and bundle different technologies, including working examples of Synergy Software, to give you a head start on your own hardware designs.

12.1   Product Examples

The Synergy Solutions come in two categories — Product Examples (PEs) and Application Examples (AEs).

Product Examples (PEs) offer a unique insight into a "design instance" of a particular end product, closely representing how the actual end product would be designed. To enable fast development of feature-rich applications, portions of the Product Examples design can be re-used in real end-products.

They come with the required hardware, Synergy Software and complete documentation, including the schematics, layouts and Gerber files, the bill of material (BOM) and special "design journey documents" describing the methodology of how and why design choices were made and the reasoning behind the selection of different components during the design process. This gives you the opportunity to modify the PE design to meet the particular needs of your application without having to re-invent the wheel.

Right now, a couple of PEs are available / planned. One of the examples is the PE-HMI1, a development board for evaluating the Synergy S7G2 Group Microcontroller (MCU) in a Human-Machine Interface (HMI) application. With its TFT LCD display and communication features like Wi-Fi 802.11a/b/g, BLE, Ethernet, USB and CAN, it closely matches the functionality and appearance of a real HMI end-product, allowing a rapid development of such products. Components included are an integrated 7-inch wide-format WVGA (800 x 480 pixels) display with a capacitive touch screen overly, a J-Link® Lite ARM® debug probe, a universal Power over Ethernet (PoE) power injector and the necessary cabling and documentation.

Figure 12-2: The PE-HMI1 provides everything needed to develop a feature rich HMI application

Another PE is the PE-DAQ1 data acquisition example based on the Synergy S3 Series MCU representing a commercial or consumer data acquisition product. It is taking advantage of the S37A's large memory and its precision analogue interfaces. Multiple sensors for motion, pressure, temperature or ambient light, together with a custom segment display and capacitive touch buttons provide a head start for your own application. For updated information about other PEs, please check out Synergy Xplorer at **http://synergyxplorer.renesas.com**.

12.2   Application Examples

The intention of Application Examples (AEs) is to demonstrate complete applications and to provide technology building-block examples that you can use to build your application upon. They include multiple components, creating a showcase of the use of multiple technologies surrounding and including the Synergy Platform.

For example, the AE of a cloud connected system for the Internet of Things (IoT) would demonstrate the use of different wireless networks like Wi-Fi, Bluetooth® low energy (BLE) Low Energy and an IEEE-802.15.4 6 LoW-PAN mesh network, as well as a HMI and cloud connectivity and services.

Another example, which is already available is the AE-CAP1 for capacitive touch. It comes with two CPU boards, one with an Synergy S124 Group MCU S124 microcontroller and one with an S3A7, and three touch sensor boards. All three will work with both CPU boards. It is certified to IEC 61000 and is delivered with the Capacitive Touch Workbench software to tune the sensitivity of the touch sensors.

Figure 12-3: The AE-CAP1 provides everything needed to develop a touch sensor application

AEs are fully documented with an application note and include the related SSP-based e^2 studio software projects. All this provides you with the information required to jump-start your own application. Up-to-date information about the latest AEs can be found at **http://synergyxplorer.renesas.com**.

12.3   More Details on the Synergy Gallery

While we talked about the Synergy Gallery already quite a bit, there is still more to it. Remember that Renesas is your point of contact for everything inside the Synergy Platform. Under *Support* you will find all what is needed: Access to the chat function for first level support by application engineers from Renesas, available for 24 hours on workdays, the Knowledge Base, which is the home for all Synergy Platform-related design help content and the Synergy Forum hosted on Renesas Rulz. Links to other support options are provided there as well.

Two other possibilities to speed up your design can be found under *Software Add-Ons* on the Synergy Gallery. The Software Add-on program provides a variety of speciality software components for developers using the Synergy Platform. These components cover multiply technology areas including extended security and functional safety, cloud services and specialized control algorithms.

Software Add-ons come in four different flavours: Renesas Qualified Software Add-Ons, Renesas Verified Software Add-Ons, Partner Showcases with Partner Projects and additional add-ons provided by Renesas directly. All of them bring even more ease of use to the Renesas Synergy™ Platform.

12.3.1 Qualified Software Add-On

Qualified Software Add-Ons (QSAs) augment the Synergy Software Package (SSP) with specialized software functionalities like purpose built connectivity stacks, security services, or control algorithms. They are sold and licensed separately from the SSP and are serviced by Renesas through the Synergy Gallery. Components of the QSAs have been sourced from Renesas and have been developed and tested under the same rigorous quality requirements as the SSP and are highly integrated into and optimized for the SSP and its API structure.

Evaluation versions of QSA software components can be downloaded from Synergy Gallery free of charge.

12.3.2 Verified Software Add-On

Verified Software Add-On (VSA) components are sold and serviced by third parties as part of the Synergy ecosystem and they are available through the Synergy Gallery. VSAs include a selection of speciality software components, for example for industrial communication protocol stacks, audio codecs, motor control applications and more.

They are verified by Renesas to ensure the compatibility with the SSP and evaluation versions can be downloaded from the Synergy Gallery free of charge. Commercial licenses have to be acquired from the VSA software vendor directly.

12.3.3 Synergy Partner Showcases

The Synergy Partner Showcase category can be accessed directly from the Synergy Gallery and contains so called Partner Projects (PP). These PPs are developed by Renesas partner companies and are demonstrated on the Synergy Platform.

They represent a wide range of products and services offered by and supported through Renesas' partner companies, providing quick and easy evaluation projects for developers to build on.

12.3.4 Additional Software from Renesas

Not only Renesas' partner companies provide additional software for the Synergy Platform, but also Renesas itself does. These add-ons to the SSP can be accessed by clicking on the SSP Utilities tab on the SSP page inside the Gallery.

One example for these add-ons is the Synergy Wi-Fi Application Framework for the Synergy Starter Kit SK-S7G2. It demonstrates how Wi-Fi connectivity can be easily added to a complex multi-threaded application using the SSP and the framework. Other add-ons, like for example one for the bootloader, will follow shortly.

The future will bring more additions to the platform, so check out the Synergy Gallery often!

Points to take away from this chapter:

- Product Examples are representations of how an actual end product would be designed.
- Application Examples are technology building-block examples you can use to build your application.
- The Synergy Gallery is your single point of access for support for all components of the Synergy Platform.
- QSAs are additional and specialized software components qualified and supported by Renesas.
- VSAs are additional and specialized software components created, licensed and supported by the third parties, which are verified by Renesas to work with the SSP.
- Even more software is provided by Renesas and their partner companies as add-ons or Synergy Partner Projects.

13 SUMMARY AND ACKNOWLEDGEMENTS

You made it! This is the last, and quite short, chapter in the book about the basics of the Renesas Synergy™ Platform. I hope you enjoyed learning about this great platform as much as I did writing about it.

With its ease of use, its low entry barriers and the complete ecosystem, consisting of production-grade software, newly designed microcontrollers, the Gallery acting assingle access point, tools and kits matching the actual needs of the designers, and solutions which resemble real end-applications and together with the great support structure, it is certainly one of the greatest revolutions I experienced during the 23 years plus I work in this industry. It is a bold new way to create tomorrow's demanding, featurerich and highly interconnected applications and developers will embrace it.

I could not imagine this book, which not only comprises the book itself, but also the web-based content and the video tutorials, becoming a reality without the help and support from all the fine people at Renesas Electronics Europe.

A special thanks from the bottom of my heart goes to the following people:

Nelly E.

for her commitment and tireless project management.

Andy H.

for giving me the opportunity to be part of this great project.

Karol S. & Jakub J.

for reproducing every single step of the exercises, making sure that they will work.

Ian H. & Giancarlo P.

for taking their time to work through a couple of the technical chapters and providing
substantial feedback.

Steve N.

for making my English understandable to readers.

Klaus R. & Britta W.

for supporting the design of the book.

Without them, their spirit, their willingness to share ideas, their reviews for technical and language accuracy, this book
would never have been possible!

ABOUT THE AUTHOR:

Richard Oed has worked as Field Applications and Systems Engineer for more than 23 years in the field of Embedded Processing,
providing support and writing software forDSPs, microcontrollers and microprocessors, as well as for data converters. He nowworks
as freelance author, writer and journalist. Richard is co-inventor of three patents and is member of the IEEE, the ACM and the VDE.

INDEX

Before purchasing or using any Renesas Electronics products listed herein, please refer to the latest product manual and/or data sheet in advance.

Renesas Electronics Europe

www.renesas.com

SAME-DAY SHIPPING on orders entered by **8:00pm** CENTRAL · NEXT-DAY DELIVERY

Digi-Key
ELECTRONICS

THE WORLD'S LARGEST SELECTION OF ELECTRONIC COMPONENTS
AVAILABLE FOR IMMEDIATE SHIPMENT®

NOW more than 6.5 MILLION PRODUCTS from over 650 SUPPLIERS

DIGIKEY.COM

ecia MEMBER · ecsn member · CEDA